VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
EKONOMICKÁ FAKULTA

KATEDRA APLIKOVANÉ INFORMATIKY

Testování webových aplikací

Testing of Web Applications

Student: Tereza Římanová

Vedoucí bakalářské práce: Ing. Jan Ministr, PhD.

Ostrava 2014

VŠB - Technická univerzita Ostrava
Ekonomická fakulta
Katedra aplikované informatiky

Zadání bakalářské práce

Student: **Tereza Římanová**

Studijní program: B6209 Systémové inženýrství a informatika

Studijní obor: 6209R001 Aplikovaná informatika

Téma: Testování webových aplikací
Testing of Web Applications

Zásady pro vypracování:

1. Úvod
2. Teoretická a metodická východiska testování webových aplikací
3. Analýza stávajícího procesu testování
4. Návrh na inovaci a implementaci procesu testování webových aplikací
5. Závěr

Seznam použité literatury
Seznam zkratek
Prohlášení o využití výsledků bakalářské práce
Seznam příloh
Přílohy

Seznam doporučené odborné literatury:

POPESKO, Boris. *Moderní metody řízení nákladů: jak dosáhnout efektivního vynakládání nákladů a jejich snížení*. Praha: Grada, 2009. ISBN 978-80-247-2974-9.
KOMZÁK, Tomáš. *Řízení IT projektů pro úplné začátečníky*. Brno: Computer Press, 2013. ISBN 978-80-251-3791-8.
ASH, Lydia. *The Web testing companion: the insider's guide to efficient and effective tests*. Indianapolis: Wiley, 2003. ISBN 04-714-3021-8.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Ministr, Ph.D.**

Datum zadání: 22.11.2013
Datum odevzdání: 09.05.2014

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * OSTRAVA * Ekonomická fakulta 1

Ing. Petr Rozehnal, Ph.D.
vedoucí katedry

prof. Dr. Ing. Dana Dluhošová
děkanka fakulty

Čestné prohlášení

Prohlašuji, že jsem celou práci, včetně všech příloh, vypracovala samostatně.

V Ostravě dne 8. 5. 2014

…….....

Tereza Římanová

Poděkování

Děkuji vedoucímu této bakalářské práce Ing. Janu Ministrovi, Ph. D. za cenné rady, připomínky a metodické vedení, čímž přispěl k jejímu vypracování.

Obsah

1 Úvod

Softwarové testování je nedílnou součástí všech stádií vývoje softwaru. Je založené na profesionalitě a preciznosti vývojového a testovacího týmu. Některé zdroje popisují testování pouze jako dynamickou kontrolu toho, že chování programu odpovídá specifikaci. V širším pohledu je to jakákoliv aktivita odhadující, že chování programu není ve shodě se specifikací. Nejobecněji se však testování považuje za samostatnou náročnou disciplínu, jejímž úkolem je ověřování kvality. S rozmachem webových aplikací ve všech podobách vzniká také zvyšující se náročnost klientských požadavků. Úkolem vývojářů a testerů je vytvořit produkt, odpovídající kvalitě a požadavkům klienta. Bez testování bychom si nemohli být jisti kvalitou vyvíjeného produktu, jejímž důsledkem jsou konkurenční výhody, obchodní úspěšnost a dobré jméno firmy. Uvolnění verze softwaru obsahující takové množství chyb, které není očekáváno zákazníkem, může vést k růstu nejrůznějších rizik. Nejčastěji se jedná o rizika spojené s finanční ztrátou ve formě pokut a poškození reputace, vedoucí k ukončení spolupráce se stávajícími zákazníky, obchodními partnery, ale také zaměstnanci, jenž nechtějí být spojováni s nekvalitní firmou. Zajištění kvality je tak pro firmu investicí, u níž se očekává návratnost v různých formách. Samotné testování však naráží na řadu chyb a problémů, kterým je třeba se věnovat.

Cílem práce je zpracování návrhu metodiky pro testování webových aplikací v malé firmě. Tato práce je rozdělena do tří částí. První se zabývá teoretickými východisky vývoje a testování webových aplikací. Druhá část analyzuje současný stav ve firmě, jejíž název je v rámci práce utajen z důvodu konkurenčního prostředí. Jsou zde zkoumány procesy testování a vývoje, včetně analýzy slabých stránek těchto procesů. V třetí části je navrhnuta inovace testovacího procesu firmy.

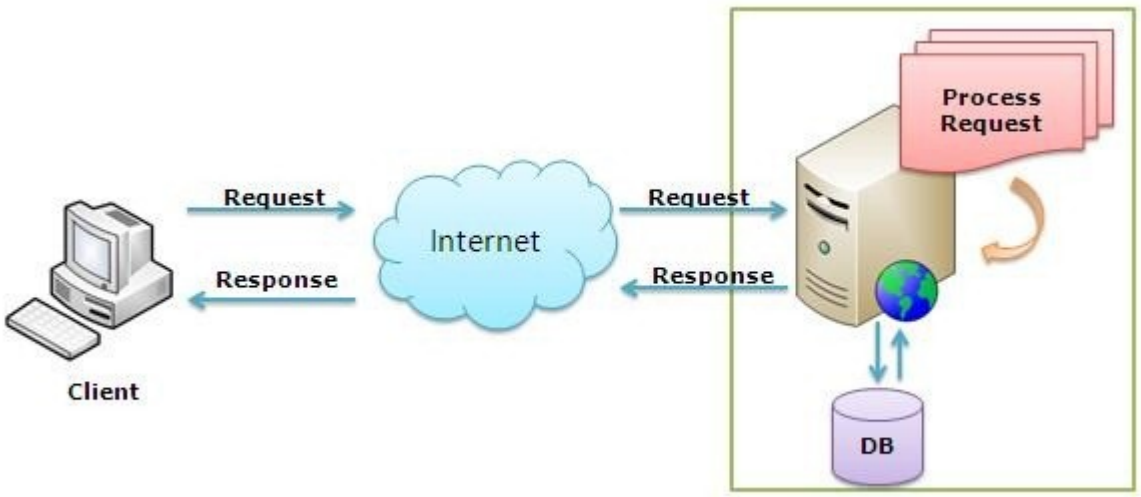
2 Teoretická a metodická východiska testování webových aplikací

2.1 Webové aplikace

Webová aplikace je konkrétní typ softwaru, který je určen pro internetové prostředí. Tato aplikace může na první pohled vypadat jako jednoduchá webová stránka, ale obvykle se jedná o složitější aplikaci provádějící netriviální úlohy a využívající databázi. Určit přesnou hranici mezi webovou stránkou a webovou aplikací je někdy nemožné. Uživatelé k webovým aplikacím umístěným na serveru přistupují pomocí internetového prohlížeče, který je dnes součástí každého operačního systému. Samozřejmostí je přístup k vždy aktuální zveřejněné verzi aplikace. Schopnost aktualizovat a spravovat webové aplikace bez nutnosti šířit a instalovat software na potenciálně tisíce uživatelských počítačů je hlavním důvodem jejich oblíbenosti. Mezi nejčastější webové aplikace patří emailové schránky, internetové obchody, online aukce, sociální sítě a další. Podstatnou výhodou je schopnost pracovat bez ohledu na typ operačního systému. V praxi ale nekonzistentní implementace kódovacích jazyků a specifikace jednotlivých prohlížečů způsobují problémy.

Struktura webových aplikací je obvykle rozložena do tří vrstev. První vrstvou je webový prohlížeč, druhou nástroje pro dynamické generování stránek. Třetí vrstvou je databáze. Webový prohlížeč posílá požadavky druhé vrstvě, a ta prostřednictvím dotazů z databáze vygeneruje uživatelské rozhraní. (Ash, 2003)

Obrázek 2.1 Schéma komunikace webové aplikace



(Zdroj obrázku: Abhijitjana, 2010)

Trendem ve webových aplikacích je poskytnutí přístupu přes web také k aplikacím, které byly dříve nabízeny jako lokální.

2.2 Technologie vývoje webových aplikací

2.2.1 HTML

Pomocí jazyka HTML se vytváří webové stránky. Tento jazyk je charakteristický používáním informací v podobě značek. Používaný termín pro tyto značky je tag. Tagy určují význam textu, který je mezi ně uzavřen. Jednotlivé tagy a jejich atributy se uzavírají do úhlových závorek např.: <*head*>. Tagy můžeme rozdělit na párové a nepárové. U párových značek je část dokumentu ohraničena dvěma tagy, vždy na konci a na začátku, přičemž koncová značka se liší od počáteční, lomítkem před názvem. Nepárové značky nevyužívají koncovou značku a slouží k jednorázovým akcím např.: <*br*> slouží k přechodu na nový řádek.

Dokument HTML může obsahovat krom HTML jazyka také komentáře, což jsou pomocné texty, které nejsou prohlížečem interpretovány. Dále se v kódu mohou vyskytnout kaskádové styly (popsány v další kapitole) a skriptovací jazyky. Struktura dokumentu HTML se skládá z několika částí:

- doctype – určení verze jazyka |HTML,
- kořenový prvek <*html*></*html*>, který musí obsahovat veškerý obsah,
- hlavička dokumentu, obsahující informace o stránce,
- tělo dokumentu, které obsahuje veškerý zobrazovaný obsah.

K vytvoření hypertextového dokumentu stačí poznámkový blok, který je součástí každého operačního systému. Existují však bezplatné editory, s funkcemi zvýraznění určitých částí syntaxe. Tyto editory dělají kód mnohem přehlednějším. Velmi oblíbený je freeware PSPad editor. Webové prohlížeče jsou programy, prezentující HTML dokument na zobrazovacím zařízení. (Janovský, 2014; Goldstein, Lazaris, Weyl, 2011)

Od 17. 12. 2012 je možné používat verzi HTML5, nabízející mimo jiné přehrávání multimédií přímo v prohlížeči, nové HTML značky a podporu offline aplikací.
(W3C, 2014)

2.2.2 CSS

Kaskádové styly (dále jen CSS) jsou metodou pro formátování HTML dokumentu, která umožňuje oddělit vzhled stránky od samotného obsahu. Tento způsob oddělení vzhledu je přehlednější a urychluje možnost změny v designu stránky. Pomocí CSS lze deklarovat barvy, velikosti, pozice a mnoho dalších vlastností jednotlivých elementů. (Kosek, 2013)
Existují tři způsoby jak aplikovat CSS v HTML dokumentu:

- přímo v textu pomocí atributu style,
- pomocí stylopisu v hlavičce stránky,
- použitím externího souboru.

Janovský (2014) tvrdí, že připojení externího CSS souboru je nejvhodnější možností, protože může být využit několika stránkami. CSS soubor je nutno připojit k HTML dokumentu pomocí speciálního odkazu.

2.2.3 Programovací jazyky

Vzhledem k tomu, že pomocí HTML a CSS nelze vytvořit složitější webovou aplikaci, je nutné využívat programovacích jazyků. Webové aplikace mohou být naprogramovány pomocí mnoha dostupných programovacích jazyků a frameworků. Nejčastějšími jazyky jsou PHP, Python, Perl, Java, Ruby a ASP.NET, přičemž nejpoužívanějším je jazyk PHP. V PHP jsou napsány i ty největší internetové projekty, včetně Wikipedie a Facebooku. Oproti tomu frameworky jsou soubory knihoven a nástrojů, které programátorům ulehčují práci v konkrétním programovacím jazyce. Tyto knihovny urychlují vývoj pomocí přehlednějšího kódu a řešení neustále se opakujících úloh. Nejdůležitějším kladem frameworků je však snížení počtu nejpravděpodobnějších chyb v aplikaci. (Láng, 2010; Majda, 2009)

2.2.4 Sémantika a přístupnost

Sémantikou se rozumí nauka o významu znaků a slov. Z hlediska webových aplikací se jedná o význam jednotlivých tagů, obecně stylistická pravidla. Každá HTML značka by měla být používána na to, k čemu byla určena. Stránky se sémantickým kódem jsou pro webové vyhledávače více viditelné a vyhledávač je bude umisťovat na vyšších pozicích ve vyhledávání, což je cílem každého vývojáře. Dalším uplatněním je využití hlasových čteček pro zrakově postižené. Tyto čtečky si umí poradit pouze se správně strukturovaným webem. (Stohwasser, 2010)

Součástí sémantického webu je struktura stránky pomocí postupné úrovně nadpisů, přičemž nadpis nejvyšší úrovně by měl být na každé stránce pouze jeden. Texty by měly být strukturovány v odstavcích a všechny netextové prvky by měly obsahovat alternativní popis. Samozřejmostí je odlišení odkazů od ostatního textu a jasné a pochopitelné ovládání webu. (Havrlant, 2005)

Pavlíček (2009) tvrdí, že: „*Za přístupný web lze dnes považovat takový web, který bude návštěvník s těžkým zdravotním postižením schopen i přes svůj zdravotní handicap, za pomoci prostředků, které má k dispozici, a způsobem, který mu vyhovuje, efektivně používat a dosáhnout svého cíle.*“. Přístupnost webu se může stát konkurenční výhodou. V současné době se zvyšuje počet seniorů a postižených uživatelů, kteří mají specifické potřeby na internetu, a je třeba se jim věnovat.

2.2.5 Responzivní design

Responzivní design je ve světě webdesignu poměrně nový pojem – poprvé ho v roce 2010 ve svém článku pro A List Apart použil americký programátor Ethan Marcotte. Zjednodušeně se jedná o způsob stylování webových stránek zaručující, že zobrazení stránky bude optimalizováno pro všechny druhy nejrůznějších zařízení, ať už se jedná o počítače, tablety nebo telefony. Díky možnostem dnešních technologií lze rozpoznat vlastnosti zařízení, na kterém je stránka prohlížena a přizpůsobit tak samotnou stránku a její obsah. Odpůrci tohoto způsobu stylování vytýkají přílišné zjednodušení webu a jeho funkcí na mobilních zařízeních. (Mareš, 2013)

2.2.6 SEO/SEM

Search Engine Marketing se zabývá zviditelněním webových stránek v internetových vyhledávačích. Můžeme jej rozdělit na placenou a neplacenou část. Neplacené části marketingu vyhledávání se říká Search Engine Optimalization (dále pouze SEO) – česky optimalizace pro vyhledávače. SEO je způsob vytváření a editace webových stránek tak, aby jejich forma a obsah byly vhodné pro automatizované zpracování v internetových vyhledávačích. Úkolem SEO je optimalizovat stránky pro algoritmy vyhledávačů a dosáhnout tak co nejlepších pozic ve výsledcích vyhledávání. SEO se zaměřuje na zvyšování čistoty a validity zdrojového kódu, na zlepšování kvality obsahu webových stránek a na růst počtu zpětných odkazů směřujících na web. Pro úspěšnou optimalizaci je potřeba provést řadu úprav přímo na webových stránkách. Základní pravidlo je, že každá stránka na optimalizovaném webu musí mít unikátní obsah. Vždy je důležité mít na paměti, že vyhledávače hodnotí

jednotlivé stránky zvlášť. Je nutné soustředit se nejen na hlavní stranu, ale i na ostatní stránky webu. Další důležitou částí je vhodný výběr klíčových slov, pod kterými budou uživatelé vyhledávat stránku ve vyhledávači a krátká neměnná URL adresa. Důležité tagy z hlediska SEO na jednotlivých stránkách:

- title – titulek shrnující obsah stránky,
- description – smysluplný popis stránky do 250 znaků,
- keywords – klíčová slova,
- nadpisy,
- popisky obrázků – vyplněné alternativní popisy. (Smička, 2004)

2.3 Metodiky vývoje

Metodikou vývoje softwaru se rozumí souhrn postupů, které využívá pracovní tým při vývoji software. Výběr vhodné metodiky, a tedy i modelu životního cyklu projektu, je klíčový pro úspěch projektu.

O důležitosti správného výběru se zmiňuje také Chapman (2011): *„složitým problémem při výběru a dodržování metodiky je činit tak s rozumem – poskytnout dostatek procesních disciplín k zajištění kvality vedoucí k obchodnímu úspěchu, ale zároveň se vyvarovat kroků, které představují ztrátu času, snižují produktivitu, demoralizují vývojáře a vytvářejí nepotřebnou administrativu.“*

V průběhu let byla vyvinuta řada schémat, z nichž má každé, své silné a slabé stránky. Tyto schémata se liší také svou vhodností pro konkrétní projekty a jejich potřeby. Mezi nejčastější tradiční modely patří Vodopádový model, Spirálový model a RUP – Rational Unified Process. (Hlava, 2011)

2.3.1 Procesní řízení

Zvyšování efektivity podnikových procesů se dnes již považuje ve většině firem za nezbytnost. Pojem procesní řízení označuje sled činností, které organizace provádí za účelem optimalizace svých klíčových procesů. Řepa (2006) definuje proces jako: „*Souhrn činností transformujících souhrn vstupů na souhrn výstupů (zboží nebo služeb) pro jiné lidi nebo procesy, používajíce k tomu lidi nebo nástroje.*“ Činnost je aktivita vykonávaná jedním pracovníkem nebo týmem v souvislém čase. (Kocourek, 2007)

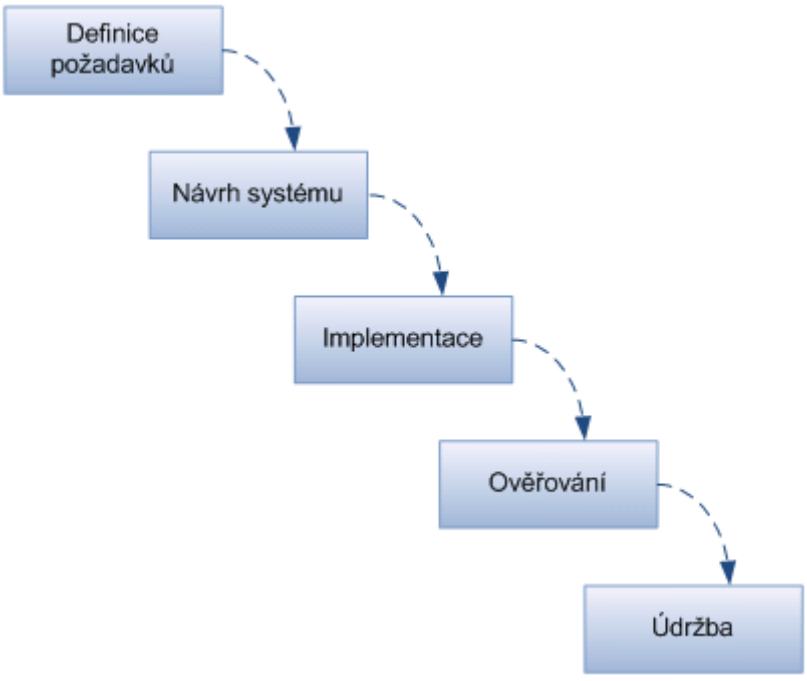
2.3.2 Vodopádový model

Nejstarší model – vodopádový – je sekvenční vývojový proces, který svou posloupností jednotlivých fází připomíná protékání vody vodopádem. Model je charakteristický tím, že vyžaduje, aby se vždy do následující fáze vstoupilo až tehdy, pokud je předchozí fáze kompletně dokončena. Pokud je počátečním fázím tvorby softwaru věnováno dostatek času, může to vést k úsporám v pozdějších fázích. Odstranění chyby, na kterou se přišlo v raných fázích modelu je levnější, než když se na tu samou chybu přijde v pozdějších fázích celého procesu. (Boehm, 1988)

Autor řady knih o softwarovém inženýrství McConnell (1996, s. 335) odhaduje, že: „*odstranění chyby v požadavcích, kterou se nepodaří odhalit až do fáze implementace nebo údržby, stojí 50 krát až 200 krát více, než kdyby se taková chyba odhalila a napravila již v etapě specifikace požadavků*.“

Hlavní nevýhodou v praktickém uplatnění modelu je nemožnosti dokončit jednu fázi a pokračovat v další, aniž by bylo nutné se k ní v budoucnu vrátit. Dalším záporem je odepření možnosti reagovat v průběhu vývoje na požadavky klienta. Vodopádový model je pro vývoj softwaru v praxi nepříliš vhodný, vzhledem k tomu, že fáze testování je až na konci celého procesu. (Hlava, 2011)

Obrázek 2.2 Vodopádový model



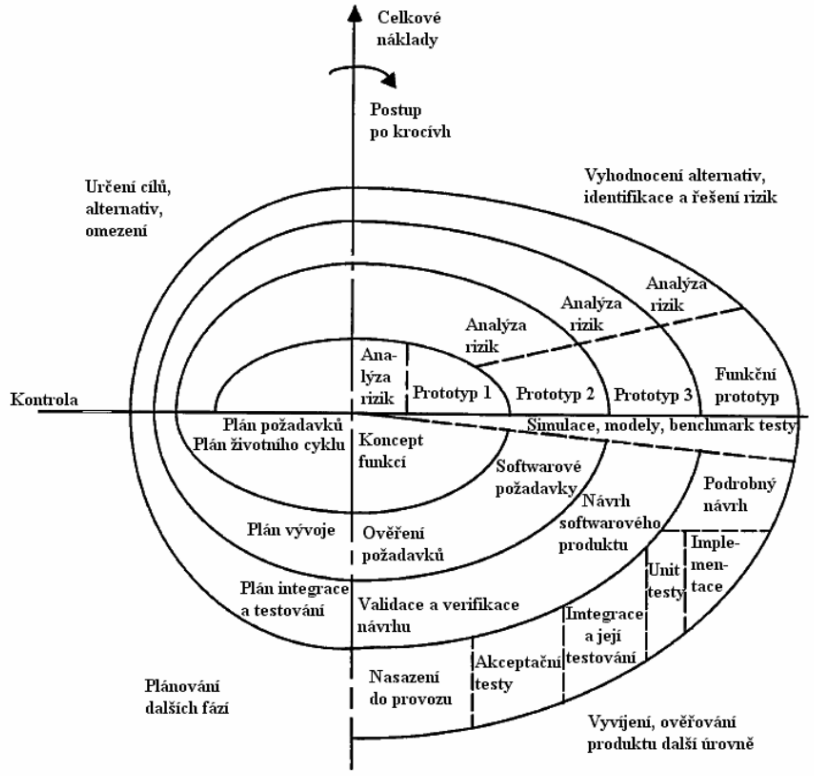
(Zdroj: VUT Brno, 2012)

2.3.3 Spirálový model

Vodopádový model popsaný v předchozí kapitole brzy po svém vzniku přestal být vyhovující. Proto softwaroví inženýři vymysleli nový přístup, odstraňující nedostatky vodopádu. Tento model funguje na principu cyklu, který obsahuje několik kroků. Tyto cykly se opakují, dokud není projekt hotov.

Spirálový model pokrývá nedostatky vodopádového modelu. Životní cyklus modelu obsahuje čtyři hlavní části – Analýzu, Vyhodnocení, Vývoj a Plánování. Po každém dokončení cyklu následuje fáze testování. Projekt je tedy testován pravidelně a díky tomu dochází k včasnému odhalení chyb, které se dají opravit v průběhu vývoje. (Boehm, 1988)

Obrázek 2.3 Spirálový model



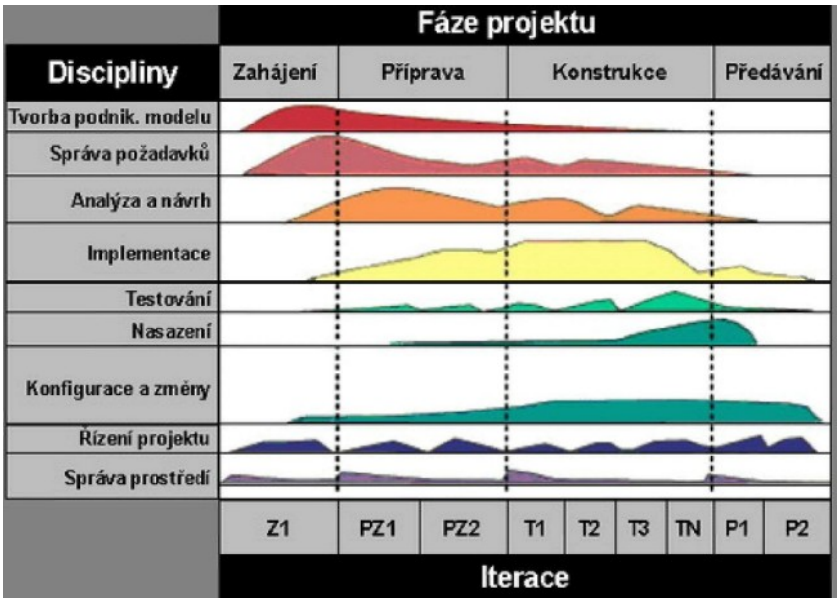
(Zdroj:Testování Softwaru, 2007)

2.3.4 RUP (Rational Unified Process)model

RUP je metodika vývoje softwaru vycházející z osvědčených postupů. Je vhodná spíše pro větší projekty, na kterých pracují větší týmy. Metodika se skládá ze čtyř fází, dále dělitelných na tzv. iterace – podmnožiny jednotlivých fází. Před započetím další fáze musí být splněna kritéria fáze předchozí. Pro modelování procesů využívá prostředků jazyka UML, což je grafický jazyk pro navrhování a dokumentaci procesů. Oblasti testování se metodika RUP věnuje velmi rozsáhle. Je na něj kladen důraz po celý životní cyklus projektu, nejvíce však

před koncem implementace a předáním konečného projektu klientovi. Díky své rozsáhlosti je RUP přece jen určen spíše větším týmům pro realizaci velkých projektů. Zvládnutí metodiky vyžaduje hlavně v úvodních fázích poměrně hluboké studium a dobře trénovaný vývojový tým. (Hlava, 2011)

Obrázek 2.4 RUP model



(Zdroj:Testování Softwaru, 2007)

2.3.5 Agilní metodiky

Cílem agilních metodik je lepší organizace práce, zapojení klienta a otevřená spolupráce. Je kladen důraz na možnost reagovat na změny požadavků v průběhu vývojového cyklu, aniž by to mělo za následek masivní přetváření již provedené práce, a tím zbytečné plýtvání časem i zdroji všech zúčastněných. Agilní metodiky vznikly jako reakce na tradiční metodiky, kterým byla vytýkána byrokratičnost, zkostnatělost a neschopnost flexibilně reagovat na změny. Agilních metodik existuje několik typů (Extrémní programování, FDD, SCRUM) a jejich společné rysy byly zformulovány v Manifestu agilního programování v USA. (Kadlec, 2004) Priority agilního programování jsou následující (Agilemanifesto, 2001):

- *„Jednotlivci a interakce před procesy a nástroji.*
- *Fungující software před vyčerpávající dokumentací.*
- *Spolupráce se zákazníkem před vyjednáváním o smlouvě.*
- *Reagování na změny před dodržováním plánu.“*

Dále bylo definováno několik principů agilního programování: (Agilemanifesto, 2001):

- *„Nejvyšší prioritou je časné a průběžné dodání SW zákazníkovi.*
- *Změny v požadavcích jsou žádoucí, neboť agilní procesy toto podporují.*
- *Dodání SW v intervalech týdnů, maximálně měsíců.*
- *Lidé musí spolupracovat denně na řešení projektu.*
- *Projekty jsou budovány kolem motivovaných jedinců, kteří mají prostředí pro odvedení dobré práce.*
- *Nejefektivnější sdělování informací je osobní konverzace.*
- *Měřítko pokroku je fungující SW.*
- *Agilní procesy podporují udržitelný rozvoj.*
- *Agilitu zvyšuje technická výjimečnost a dobrý design.*
- *Klíčová je jednoduchost, tedy maximalizovat množství nevykonané práce.*
- *Nejlepší návrhy vznikají ze samo-organizujících se týmů.*
- *Vývojový tým se pravidelně zamýšlí nad tím, jak se stát efektivnější.“*

Zatímco ve světě se agilní metodiky stále více prosazují, v české praxi se zatím nedočkaly většího rozšíření.

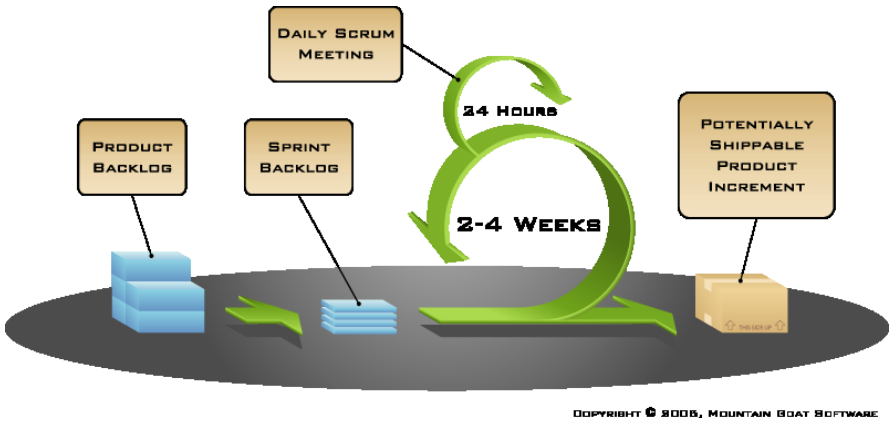
2.3.6 SCRUM

Scrum je jednou z agilních metodik řízení vývoje softwaru. Je určena pro malé pracovní týmy. Vývoj probíhá v krátkých iteracích, cyklech, které se nazývají *Sprinty*. Sprint je neustále kontrolovaná činnost, obvykle trvající 14 dní. Veškeré požadavky projektu jsou shromážděny na jednom místě, nazývající se *Product backlog*. Začátek každého sprintu začíná výběrem požadavků, které mají být po ukončení iterace funkční. Součástí Scrum je každodenní porada, shrnutí požadavků a jejich plnění. Zákazník má těsný kontakt s vývojovým týmem a na konci každé iterace má přístup k funkční verzi produktu. Během vývoje probíhá průběžné testování k ověření funkčnosti prvků z dané iterace. (Kratochvíl, 2010)

Role účastníků celého vývojového procesu se dělí do dvou skupin. První skupinou jsou Stakeholders (lidé ze strany zákazníka) a manažeři, druhou jsou Product owner (osoba zodpovídající za priority sprintu) a Scrum Master (manažer vývojářů). Tyto skupiny se zjednodušeně, podle povídky o praseti a kuřeti, nazývají Chickens a Pigs. Chickens jsou osoby, kterých se problém pouze týká a přímo nezodpovídají za vývoj, tj. manažeři

společnosti nebo uživatelé produktu. Pigs jsou osoby související přímo s vývojem, tedy součásti vývojového týmu. Denních meetingů se musí zúčastnit všechny Pigs, pro Chickens to není povinné, zúčastnit se mohou, nejsou však oprávněni diskutovat. Výhodou metody Scrum je možnost rychlé reakce na změny požadavků a celkové zrychlení vývojového procesu. (ÚVT Muni, 2014)

Obrázek 2.5 Metodika Scrum



(Zdroj: Mountain Goat Software, 2005)

2.3.7 Extrémní programování

Extrémní programování, zkráceně XP, je metoda vývoje softwaru, která vychází z principů agilního vývoje. XP je vhodné pro malé a střední firmy a je označováno jako účinný a nerizikový způsob, jak vyvíjet software. Extrémní se nazývá proto, že všechny činnosti této metodiky jsou dotaženy do extrému:

- pokud se osvědčují revize zdrojového textu, budeme jej neustále revidovat (tato myšlenka vede k tezi párového programování),
- jestliže se osvědčuje testování, budou všichni neustále testovat (testování jednotek) a testovat budou dokonce i zákazníci (testování funkcionality),
- osvědčuje-li se návrh, zařadíme jej každodenně do činností všech (refaktoring),
- pokud se osvědčuje jednoduchost, vždy ponecháme systém s nejjednodušším návrhem vyhovujícím požadovaným funkcím (to nejjednodušší, co ještě může fungovat),
- jsme-li přesvědčení o důležitosti architektury, budou ji všichni neustále definovat a propracovávat,
- jestliže se přesvědčíme o důležitosti testování integrace, budeme integrovat a testovat několikrát denně (nepřetržitá integrace),

- pokud se osvědčují krátké iterace, uděláme je opravdu krátké: vteřiny, minuty a hodiny namísto týdnů, měsíců a let.

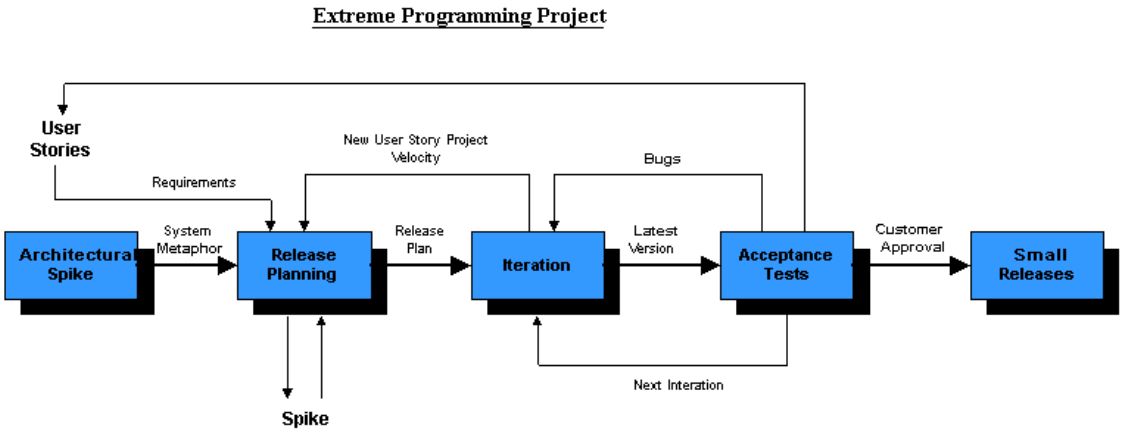
XP, znázorněno na obrázku 2.6, slibuje snížení projektových rizik, zlepšení schopnosti reagovat na změny a vylepšování produktivity během celého životního cyklu systému. Mezi nejdůležitější pojmy XP patří čtyři hodnoty, kterými se řídí a na kterých staví:

- komunikace - *XP* se zaměřuje na udržení řádných komunikačních toků tím, že využívá mnoho postupů, které nelze bez komunikace, *XP* dále definuje zvláštního člena projektu, který detekuje výpadky komunikace a znovu navazuje vztahy mezi lidmi,
- jednoduchost – XP sází na to, že je lepší udělat něco jednoduchého dnes a zítra případně zaplatit víc za případnou změnu, než platit dnes za složitou věc, která se ovšem zítra nemusí využít,
- zpětná vazba - první zpětná vazba je v řádu minut a dnů a týká se testů jednotek, které dávají okamžitou informaci o stavu nově napsané jednotky, další je v řádu týdnů a měsíců a realizuje se testy funkčnosti,
- odvaha - je-li odvaha týmu kombinována s předchozími třemi hodnotami, stává se nesmírně cennou.

Výhodou je v ideálním případě je velmi efektivní vývoj, aplikace je dodána rychle a splňuje požadavky svého zákazníka. (Beck, 2002)

Kadlec (2003) tvrdí: „*Ne každý člověk je vhodný do XP týmu. Mnozí vývojáři odmítají myšlenky párového programování, kolektivního vlastnictví kódu, permanentní komunikace. Autor XP sice uvádí, že většina si nakonec zvykne, nicméně z diskusí, které vede odborná veřejnost, je cítit silná pochybnost o tomto tvrzení*.“

Obrázek 2.6 Metoda Extrémní programování



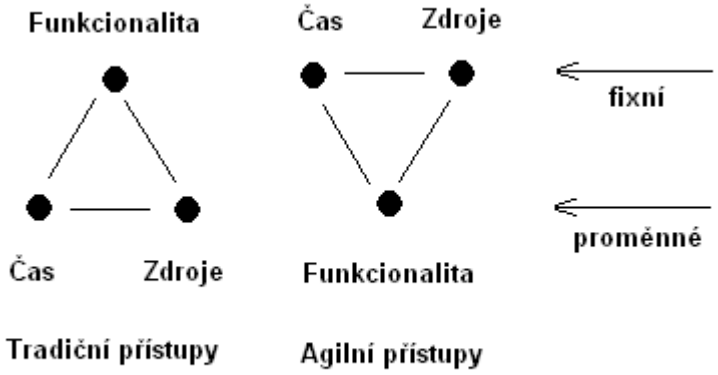
(Zdroj: RippleImpact, 2001)

2.3.8 Rozdíl mezi Agilním a tradičním způsobem vývoje

Odlišnost mezi tradičním a klasickým pojetím vývoje softwaru je naznačen na obrázku 2.7. Tradiční metodologie vyžaduje fixní specifikaci požadavků na projekt. V roli proměnných zde vystupují čas a zdroje. V praxi to znamená, že pokud je třeba dodatečně upravit specifikaci, je nutno počítat s prodloužením termínu ukončení projektu.

Agilní přístupy naopak prohlašují čas a zdroje za fixní a mění se pouze funkcionalita produktu. (Buchalcevová, 2002)

Obrázek 2.7 Rozdíl tradičního a agilního pojetí vývoje software



(Zdroj: Buchalcevová, 2002)

2.3.9 Refaktoring

Fowler (2003, str. 20) popisuje refaktorování jako: „*Proces provádění změn v softwarovém systému takovým způsobem, že nemají vliv na vnější chování kódu, ale*

vylepšují jeho vnitřní strukturu. Je to disciplinovaný způsob pročišťování kódu s minimálním rizikem vnášení chyb.“ Refaktoring ve své podstatě zlepšuje a zpřehledňuje kód softwaru až poté, co byl napsán. Jeho cílem je zlepšit srozumitelnost a ulehčit budoucí změny softwaru. Důležitou vlastností refaktoringu je to, že koncový uživatel nebo jiný programátor si nesmí všimnout, že došlo ke změně v kódu, protože funkčnost musí zůstat beze změny (výjimkou může být změna rychlosti aplikace).

V programovém kódu je vhodné používat jasné názvy proměnných a odstranit duplicity. Zjednodušit podmíněné výrazy, zkrátit dlouhé metody nebo přepsat kód, který potřebuje komentář tak, aby komentář nepotřeboval. Refaktoring by měl být prováděn před přidáním nové funkcionality do kódu, při revizích kódu nebo ve chvílích, kdy je třeba opravit nějakou chybu. Nedílnou součástí této zjednodušující činnosti je testování, neboť se jedná o úpravu kódu, která může způsobit nově vzniklé chyby.

Manažeři firmy mohou vidět refaktoring jako nevýznamnou činnost, která prodlužuje dobu projektu a zvyšuje tak náklady. Ve skutečnosti ale může ušetřit náklady na úpravy kódu a opravy chyb v budoucnu. Fowler (2003, str. 36) dodává, že: „*Kód, kterému rozumí jen počítač, dokáže napsat každý. Dobří programátoři píší kód, kterému rozumí lidé.*“

2.3.10 Vývoj řízený testy

Vývoj řízený testy (Test driven developement dále jen TDD) je klíčovou součástí extrémního programování, o kterém bylo psáno výše. Jedná se o metodu vývoje software, kdy se před samotným napsáním kódu napíše test, který ověřuje, že se aplikace chová korektně. Teprve poté se napíše samotný kód. „*Účel vytváření testů v TDD není kontrola, zda kód souhlasí se specifikací, ale testy zde představují nástroj pro návrh systému*.“ Ačkoli obliba extrémního programování jako celku postupně upadá, princip vývoje řízeného testy má šanci udržet se na výsluní delší dobu. (Fi Muni, 2005) Princip TDD je podle (Beck, 2004) shrnut do tří základních částí:

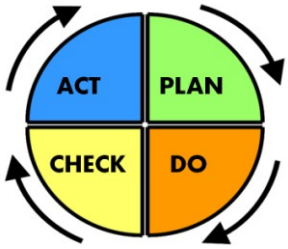
- červená – napište malý test, který nefunguje nebo nepůjde spustit,
- zelená – rychle implementujte testovanou logiku tak, aby test proběhl,
- refaktorování – zbavte se všech duplicit a upravte kód do přijatelné podoby.

Žádné studie jasně neprokázaly rozdíl mezi TDD a některou z alternativ v kvalitě nebo produktivitě. Rozdíl ale Beck (2004) popisuje ve snížené chybovosti, zlepšení pracovních vztahů v týmu a čistějším kódu.

2.3.11 PDCA

PDCA (z anglického Plan, Do, Check, Act) je model zlepšování procesů, který je založen na posloupnosti čtyř kroků (obrázek 2.8.). Tyto činnosti jsou popsány jako plánování, provedení, kontrola a akce. Cílem je prověřit aktuální stav a identifikovat problémy, realizovat řešení, zhodnotit výsledky testu a na základě vyhodnocení rozpracovat konečné řešení. Tento model má široké spektrum využití, proto je velmi oblíbený. (Sedláček, 2011)

Obrázek 2.8 Model PDCA

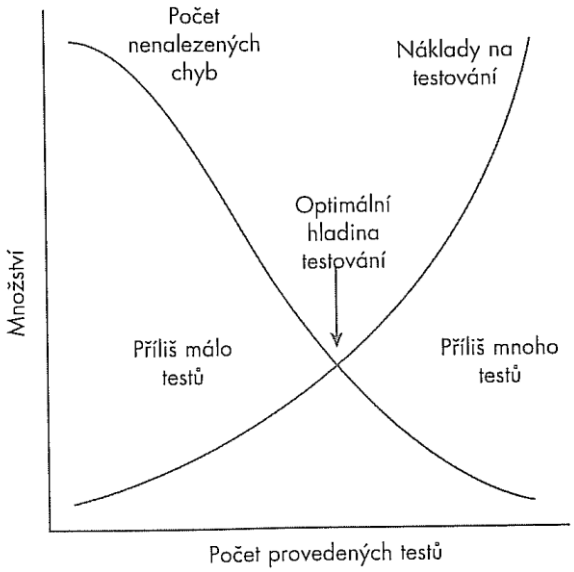


(Zdroj: Netcoach, 2011)

2.4 Testování

Testování lze podle (Patton, 2002, s. 47) definovat jako: „*potencionálně nekonečný proces, a o jeho zastavení tak musí být rozhodnuto na základě uvážení různých faktorů, jako je jeho postupně se snižující efektivita, časové a rozpočtové omezení, provedení všech klíčových testů a podobně.*“. Z tohoto vyjádření plyne, že každý testovaný projekt má optimální hladinu testování, za níž už není výhodné v testování pokračovat z různých důvodů.

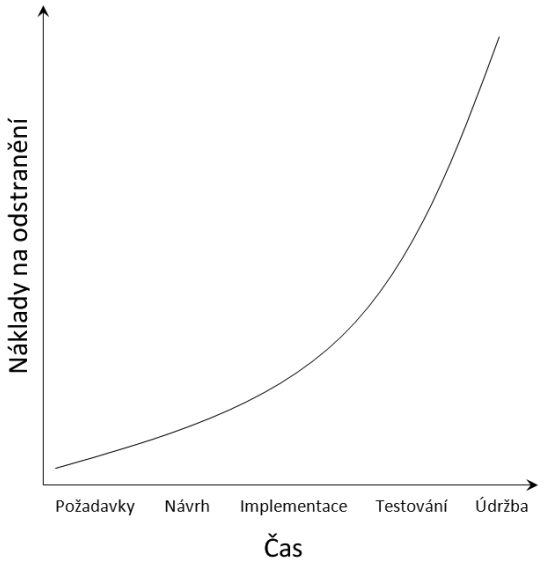
Obrázek 2.9 Optimální hladina testování



(Zdroj: Patton, 2002)

Jasným účelem testování je pak vyhledání chyb, reportování, zaručení jejich opravení, a to vše v co nejkratší době, protože mezi cenou za odstranění chyby a časem je rostoucí tendence.

Obrázek 2.10 Vztah mezi fází, ve které je defekt identifikován, a cenou za jeho odstranění



(Zdroj: Havlíčková, Roudenský 2013)

2.4.1 Axiomy testování

Axiomy jsou známé pravdy. V kontextu této práce je můžeme definovat jako pomyslná pravidla testování. Jejich znalost, zlepší celému pracovnímu týmu celkový přehled o vývoji a testování softwaru.

„Žádný program není možné otestovat kompletně.“

Počet možných vstupů, výstupů a cest, které vedou skrze software je tak obrovský, že otestovat všechny možnosti je nereálné.

„Testování softwaru je postavené na riziku.“

Každý tester musí zúžit množinu možných testů na podmnožinu, kterou je schopen vykonat, přičemž podstupuje riziko, že přehlédne chybu, která se objeví až po předání zákazníkovi. Tedy v té nejhorší fázi pro odhalení chyby a je nucen akceptovat následky.

„Testování nikdy nemůže prokázat, že chyby neexistují.“

Je možné provádět jakékoliv množství testů, ale v žádném okamžiku není možné zaručit, že žádné další chyby nebudou nalezeny. Dalším postupem je pokračování v testování a možné nalezení dalších chyb.

„Čím více chyb najdete, tím více chyb v softwaru je.“

Stává se, že tester dlouho nenajde žádnou chybu a zanedlouho jich najde hned několik najednou. Důvodem může být špatný den programátora nebo jeho slabina. Často je na vině problém v návrhu nebo architektuře softwaru.

„Paradox pesticidů.“

Pokud se proces testování v každé iteraci opakuje beze změny v testech, je pravděpodobné, že chyby sice budou odstraněny, ale na mnoho chyb, které nejsou pokryté testy, se vůbec nepřijde. Proto je pro nalezení dalších chyb nutné testy obměňovat a prověřovat různé části programu.

„Ne všechny nalezené chyby se opraví.“

Ani dobře provedená a pečlivá práce testera nezaručí, že všechny jím nalezené chyby budou opraveny. V nejhorším případě na opravu chyb není dostatek času nebo je oprava ve složitém systému příliš riskantní. Chyby v málo používaných funkcích se dají jednoduše opominout a neopravovat nebo se zjistí, že chyba vlastně není chybou, ale novou funkčností.

„Je těžké říct, kdy je chyba chybou.“

Chybou se nazývá funkčnost neodpovídající specifikaci. Je to činnost, kterou by software dělat neměl, ačkoliv ji dělá. Z druhého pohledu je to činnost, kterou by software dělat měl, ale funkčnost chybí.

kterou software dělá, ale dělat by neměl, nebo opačně nedělá, ale měl by dělat. Rozpoznání chyb záleží na konkrétním testerovi, jeho uvážení a také projektové specifikaci.

„Specifikace produktu nejsou nikdy konečné.“

Softwarový tester musí předpokládat, že se specifikace bude v průběhu projektu měnit. Přidají se nové funkce nebo se odstraní funkce již otestované. V takovém případě je nutné zůstat flexibilní a kontrolovat projekt se vždy aktuální specifikací.

„Testeři nejsou těmi nejoblíbenějšími členy týmu.“

Vzhledem k tomu, že testerovou pracovní náplní je vyhledávání problémů v práci svých kolegů, občas může docházet ke konfliktům na pracovišti. Řešením je oznamovat nejen špatné zprávy, ale i dobré. Pochválit část programu bez chyb nebo včasné opravení, rozhodně zvedne náladu v kolektivu.

„Testování je přesná technická disciplína.“

S růstem softwarového průmyslu se z testování stává technická disciplína, která vyžaduje specializované pracovníky. Na trhu práce roste zájem o softwarové testy, protože vyvíjet software s chybami je v dnešní době velmi nevýhodné a drahé. (Patton, 2002)

2.4.2 Metody a typy testování

Jednotlivé typy testování často provádí různí lidé ve vývojovém týmu a na testování je nahlíženo z různých úhlů pohledu.

Automatizace testování

Automatizace testování softwaru je v praxi nepříliš obvyklou disciplínou, přestože v některých případech přináší výrazné úspory prostředků a zvýšení kvality výsledného produktu. Automatizace má za cíl usnadnit a zefektivnit postup testování s co nejmenším množstvím zásahu lidského faktoru. U aplikací s životností v řádech měsíců se nevyplatí vytvářet automatizované testy, naopak u aplikací s delší životností ano. Dalším faktorem při realizaci těchto testů jsou časté změny v aplikaci, které by znamenaly častou aktualizaci automatizovaných testů. Pokud tester prochází neustále ty stejné testovací scénáře, zvyšuje se pravděpodobnost, že může některou chybu přehlédnout. Využití automatizace tento nepříznivý jev zcela odstraňuje. Tyto testy ovšem nemůžou z hlediska kvality zcela nahradit manuální testování, proto je automatizace brána jako pomůcka k prověřování softwaru. Na trhu se objevuje několik dostupných nástrojů pro automatizované testování ať už komerčních nebo nekomerčních. Jak tvrdí Havlíčková a Roudenský (2013, str. 56): „*Automatizace v současné situaci není nástupcem manuálního testování, je však jeho suplementární aktivitou, která v závislosti na povaze projektu může a nemusí být efektivnější.*“

Vývojářské testování

Testování kódu programátorem se provádí jako první, bezprostředně po jeho napsání. Toto testování se zaměřuje na menší části kódu, zjišťuje se, jestli kód dělá to, co programátor zamýšlel a jestli vše funguje správně. Úkolem testů je tedy kontrola funkčnosti všech uměle vytvořených situací, které by mohly v ostrém provozu aplikace nastat. V této fázi testování je z hlediska nákladů oprava chyb nejméně nákladná, ale schopnost rozpoznat chyby, které mohou nastat integrací, je zde omezená. (Hlava, 2011)

Integrační testování

Tento typ testování je zaměřen na kontrolu systému jako celku, skládající se z více komponent softwaru tzv. unitů. Testuje se vzájemná komunikace v rámci aplikace a také propojení s okolím. V této fázi se objevují chyby, na které většinou nebylo možné narazit při testování jednotlivých komponent aplikace. Odhalení a opravení těchto chyb je velmi důležité jak z hlediska dalšího vývoje aplikace, tak z hlediska nákladů na vývoj. (Hlava, 2011)

Akceptační testování

Pokud všechny předchozí etapy testů proběhly bez větších nedostatků, je možné předat aplikaci zákazníkovi. Akceptační testy obvykle provádí zákazník, aby si ověřil, že projekt je podle jeho představ a může být nasazen do ostrého provozu. Kontroluje se tedy splnění požadavků ve specifikaci projektu a bezchybná funkčnost. V této fázi je odhalení a oprava jakýchkoliv chyb nepříjemnou zátěží pro obě strany. Zkušenosti z praxe nám říkají, že i v této fázi, kdy je projekt téměř odevzdán, si zákazník může přát další změny v projektu, které naruší časový plán vývoje projektu a je nutno znovu projít všechny fáze testování. (Hlava, 2011)

Verifikace a validace

Verifikace a validace jsou nezbytnou součástí každého projektu a testování nám pomáhá ověřit tyto vlastnosti. Verifikace je ověření, že software správně implementoval funkce, které byly upřesněny ve specifikaci projektu. Klade si otázku, zda funguje správně to, co jsme vytvořili. Oproti tomu cílem validace je ověření, že software odpovídá požadavkům klienta, tedy zda jsme vytvořili to, co zákazník chtěl. (Šarmanová, 2007)

Statické a dynamické testování

Zdali se jedná o statické nebo dynamické testování se posuzuje podle toho, zda je pro testování nutné spuštění software. Vzhledem k tomu, že statické testování nevyžaduje běh softwaru, je možné s ním začít ještě před vytvořením prvního spustitelného prototypu. Do statického testování řadíme kontrolu specifikace požadavků a procházení zdrojového kódu.

Dynamické testování je založeno na spustitelné verzi softwaru a kontroluje vstupy, výstupy a interakce softwaru s uživatelem. (Borovcová, 2009)

Černá a bílá skřínka

Testování, které vychází z informací o dané aplikaci, nazýváme testováním černé a bílé skřínky. Jako černou skřínku označujeme stav, kdy tester nevidí vnitřní logiku aplikace, ale pouze vstupy a výstupy. Smyslem tohoto testování je zjistit chování softwaru z uživatelského hlediska. Tento typ je častější, protože nevyžaduje od testerů znalost programovacího jazyka. Při testování bílé skřínky má tester přístup ke zdrojovému kódu, může si odvodit souvislosti fungování aplikace a dává mu to možnost otestovat situace, které nejsou z vnějšího pohledu patrné. Nevýhodou je, že se tester nedokáže vcítit do pozice

běžného uživatele, která je velmi důležitá. Pro úplnost vznikla mezi těmito dvěma typy třetí kategorie nazvaná testování šedé skřínky. Šedou skřínku můžeme definovat tak, že tester má kromě znalostí vstupů a výstupů i základní znalosti o vnitřních procesech aplikace. Tato situace bývá v praxi nejčastější. Informace, které tester má, nejsou tak hluboké, aby se dalo mluvit o bílé skřínce, ale již zasahují za pomyslnou hranici skřínky černé. (Borovcová, 2009)

Funkční testování

Funkční testování se zaměřuje na ověřování funkčnosti aplikace na základě očekávaného chování definovaného ve funkčních požadavcích klienta. Všechny stavy, které jsou v aplikaci implementovány, jsou procházeny testerem, který simuluje chování koncového uživatele. Na tuto kategorii testování je kladen velký důraz po celou dobu vývojového cyklu, protože přináší velké množství odhalených chyb ve srovnání s ostatními kategoriemi. (Jorgensen, 2008)

Za tzv. nefunkční testování se někdy považuje kontrola požadavků, které přímo nesouvisí s funkcemi aplikace, tedy testování výkonnosti, použitelnosti a bezpečnosti. Tyto další vlastnosti budou definovány v další části této práce. (Borovcová, 2009)

Zátěžové testování

Páral (2010) definuje výkon webových aplikací: "*Jedním z nejdůležitějších kritérií úspěšnosti. Pokud je aplikace rychlá, má šanci na úspěch. Pokud je aplikace pomalá, je velmi pravděpodobné, že bude rychle zapomenuta.*". Zátěžové testy simulují práci několika uživatelů, při které se zvyšují nároky na hardware aplikace. Výkon stroje, na kterém aplikace běží, můžeme sledovat a dále analyzovat a řešit výkonnostní problémy. Pro generování simulované zátěže můžeme využít dostupných softwarových nástrojů pro měření výkonnosti a vytváření zátěže. Nejpoužívanějším nástrojem je bezplatný program JMeter – Java aplikace s grafickým rozhraním, ve které si tester může nastavit, kolik uživatelů bude posílat požadavky, na jaké objekty apod. Před samotným testováním je důležité stanovit si kritéria a cíle zátěžových testů. Obvykle nás zajímá maximální počet současně přistupujících uživatelů k aplikaci, doba odezvy, ale také jak rychle se aplikace vypořádá s výpadkem serveru. (Hlava, 2011)

Zátěžové testy lze rozdělit na jednotlivé dílčí typy, podle toho, co nám ověřují:

- test hraniční zátěže – nalezení maximálního počtu procesů, při kterém aplikace selže,
- výkonnostní test – zatížení aplikace a měření její vlastností při zátěži,
- test odolnosti – objevení nedostatků při nepřetržitém provozu aplikace,
- test selhání – co se stane, když aplikace selže,
- test části infrastruktury – zjištění nejslabšího článku aplikace,
- test citlivosti sítě – testování rychlosti a spolehlivosti sítě,
- test objemu dat – ověření chování aplikace při zvyšujícím se objemu dat.

Zátěžové testování webových aplikací má smysl, pokud se chceme vyhnout fatálním následkům, které při provozu aplikace mohou nastat. (Lukeš, 2014)

Uživatelské testování použitelnosti

Definici použitelnosti webu nejlépe vystihl Štrupl (2010): *„Použitelný je takový web, který se návštěvníkům dobře používá, kde se dobře orientují a rychle naleznou, co hledají. Kde se neztrácí, nedělají zbytečné chyby.“*. Díky uživatelskému testování lze zjistit, které prvky aplikace nejsou pro uživatele srozumitelné. Toto testování je vhodné provádět v jakékoliv části vývoje. Opět zde platí, čím dříve se na problém přijde, tím rychleji a levněji se problémy odstraní. Průběh uživatelského testování se skládá z několika kroků:

- analýza cílových skupin aplikace a jejich potřeb,
- vytvoření testovacího scénáře z jednotlivých úkonů, které simulují chování reálného uživatele,
- výběr testerů, kteří budou odpovídat cílovým skupinám návštěvníků,
- testování pod dohledem odborníka, který zaznamenává veškerá zaváhání a problémy,
- analýza a vyhodnocení testování, návrh vhodných změn.

Na základě uživatelského testování lze zjistit vnímání aplikace návštěvníky. Ověří se srozumitelnost textů, bezproblémová orientace napříč aplikací, příjemná grafika, rychlost odezvy a mnoho dalších vlastností. Na českém trhu najdeme i několik firem, které se specializují na pokročilé uživatelské testování. Tyto firmy nabízejí např. testování oční kamerou, která zaznamenává, kam se uživatel na webu dívá, nebo teplotní mapy webu, které zaznamenávají pohyby kurzoru myši. (Štrupl, 2010)

Testování bezpečnosti

Nejen ve webových aplikacích, ale v softwaru obecně se vyskytuje množství tzv. zranitelností, což jsou stavy softwaru, které mohou být zneužity útočníkem k ovlivnění bezpečnosti. Hlavním cílem bezpečnostního testování je zjištění těchto rizikových stavů a jejich minimalizace případně úplné odstranění. Častým cílem útočníků je zjištění citlivých informací z aplikace např. informace o uživatelích, jejich kontaktní údaje a hesla, finanční informace firmy, know-how apod. Bezpečnostní hrozby ve webových aplikacích můžeme rozdělit na útoky proti uživatelům a útoky proti samotné aplikaci. (Kümmel, 2014)

S testováním bezpečnosti mohou pomoci bezpečnostní skenery, což jsou nástroje prohledávající aplikaci, zkoušející obvyklé slabiny.

Testování kompatibility

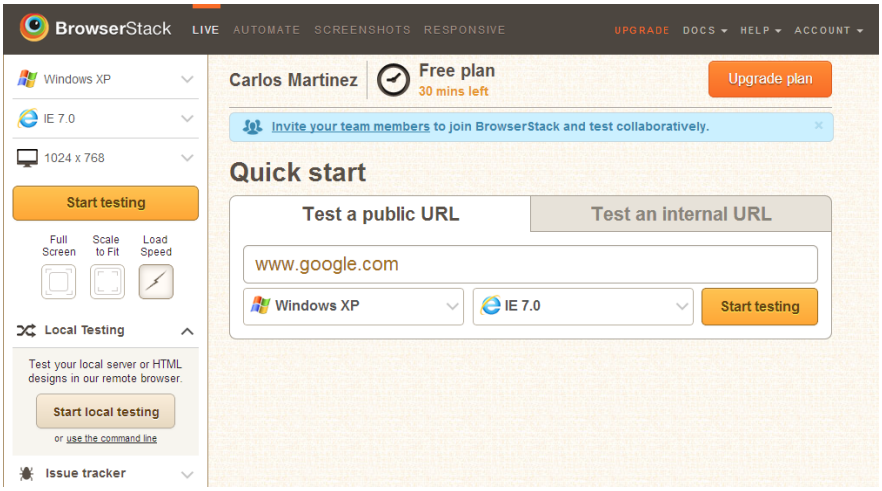
Webové aplikace jsou zobrazovány v internetových prohlížečích, přičemž každý prohlížeč může kód aplikace interpretovat mírně odlišným způsobem. Cílem při vývoji webové aplikace je tyto odlišnosti eliminovat a zajistit tím tzv. kompatibilitu. Obecně existují tři přístupy ke kompatibilitě:

- optimalizovat aplikaci pro konkrétní prohlížeč a na odlišnosti v jiných nedbat,
- nepoužívat sporné prvky a spokojit se s obsahově chudší, ale funkční aplikací,
- zjišťovat typ a verzi prohlížeče a podle něj načíst funkční aplikaci. (Ash, 2013)

Aplikace je vhodné optimalizovat pro pět základních internetových prohlížečů a různé jejich verze. Těmi nejpoužívanějšími (seřazeny podle počtu uživatelů) jsou Google Chrome, Mozilla Firefox, Internet Explorer, Safari a Opera. (w3schools, 2014)

Existují určité standardy pro HTML a CSS, ale prohlížeče je ne vždy respektují. Na internetu jsou k dispozici online aplikace, které dokážou nasimulovat aplikace na různých platformách, v různých prohlížečích a dokonce i různých verzích. Například aplikace BrowerStack nabízí cloudovou službu pro testování webových stránek. Tato služba nabízí komplexní testování webové stránky na nejrůznějších verzích internetových prohlížečů a operačních systémů, včetně zobrazení toho, jak daná stránka vypadá.

Obrázek 2.11 Aplikace BrowerStack pro testování kompatibility



(Zdroj: Martinez, 2014)

Pouze simulátory nestačí, pokud chce firma opravdu responzivní design:

- nedá se jim stoprocentně věřit,
- klikání v simulátoru nenahradí pohyb palcem na displayi,
- simulátory nenasimulují problémy s výkonností.

Bez reálných zařízení nelze vyladit web zcela spolehlivě vzhledem k velkému počtu možného hardware. Pro ty firmy, které nechtějí investovat do zařízení na testování, existuje služba na vypůjčení všech základních zařízení. (Michálek, 2013)

Porovnání testovacích přístupů

V následující tabulce 2.1. jsou porovnány jednotlivé testovací přístupy. U každého je uvedena cena testování, tj. množství nákladů potřebných na typ testování, flexibilita testů ke změnám ve specifikaci, nutnost potřeby odborných znalostí pro testování a cena chyby.

Tabulka 2.1 Porovnání testovacích přístupů

typ testování	cena testování	flexibilita ke změnám ve specifikaci	nutnost odborné znalosti	cena chyby
automatizované testování	vysoká	malá	velká	střední
vývojářské testování	střední	velká	střední	nízká
integrační testování	střední	malá	střední	vysoká
akceptační testování	střední	malá	nízká	vysoká
funkční testování	střední	malá	nízká	vysoká
zátěžové testování	střední	velká	střední	vysoká
uživatelské testování	střední	střední	nízká	střední
testování bezpečnosti	střední	velká	velká	vysoká
testování kompatibility	střední	střední	střední	střední

(Zdroj: vlastní)

2.5 Nástroje pro usnadnění práce při testování

2.5.1 Firebug

Firebug je rozšíření pro internetový prohlížeč Mozilla Firefox. Tento nástroj slouží k pohodlnému a rychlému ladění webových stránek. Nabízí funkce jako: kontrola a editace HTML a CSS kódu, sledování dotazů vyvolaných stránkou, ladění JavaScriptu a hlavně okno pro zobrazení chyb, které vzniknou při zobrazení stránky. Hlavně poslední funkce je pro testery vhodnou pomůckou. Dále usnadňuje práci hlavně programátorům při ladění chyb. (Daněk, 2007)

2.5.2 Developer tools v Internet Explorer

Vývojářské nástroje v Internet Explorer poskytují velké množství informací o tom, jak prohlížeč vykresluje webové stránky. Nástroj obsahuje funkce, které nabízí výše zmíněný Firebug, ale má k dispozici i něco navíc. Optimalizace webu se neobejde bez měření toku v síti. Developer tools nabízí přehled požadavků i odpovědí, měření odezvy uživatelského rozhraní nebo měření spotřeby paměti. Pro testování webové stránky napříč prohlížeči existuje funkce emulace, která umí nasimulovat vykreslování v různých verzích Internet Exploreru, zobrazení na mobilním zařízení a také lze přepínat mezi zobrazením na výšku či šířku. (Jahoda, 2013)

2.5.3 ModernIE

Modern.ie je webová aplikace, kterou spustila firma Microsoft. Zaměřuje se testování webových stránek v Internet Exloreru. *"Víme, že uživatelé starších verzí IE jsou při testování webových stránek opravdovým oříškem,"* uvedl na blogu Gavin (2013). *"Chceme pomoci. Chceme, aby se web posunul kupředu. A opravdu chceme, aby vývojáři strávili více času inovováním a méně času testováním.*" dodal.

Možností testování je několik, nejjednodušší je volba *Scan a webpage*, kde stačí zadat adresu stránky. Stránka je následně otestována na následující položky:

- známé problémy s kompatibilitou - seznam nejčastějších chyb, které způsobují problémy s kompatibilitou napříč prohlížeči,
- mód kompatibility – detekuje kód, který způsobuje problémy s prohlížením stránky v IE verzi 9 a IE verzi 10,
- frameworky a knihovny – test prozkoumává knihovny, které způsobují nejvíc problémů z hlediska kompatibility,
- Web Standards Docmode – zkoumá, zda stránka dává prohlížeči informace o použití nových webových standardů typu HTML5 a CSS3,
- CSS prefixy – test zkoumá, zda na stránce nechybí specifické prefixy, jejichž absence má vliv na kvalitu zobrazení v různých prohlížečích,
- plug-iny – kontroluje, zda stránka vyžaduje přítomnost plug-inu v prohlížeči,
- responsive web-design – kontroluje, zda stránka využívá media queries a umožňuje tak stejnou kvalitu zobrazení na malém tabletu, stejně jako na 24" monitoru,
- detekce prohlížeče – kontroluje, zda daná stránka umí detekovat klientský prohlížeč a podle toho nastavit optimální zobrazení,
- prohlížení dotykem – kontroluje, zda je stránka optimalizována pro prohlížení na zařízeních, jež jsou ovládána dotykem,
- Start Screen Site Title – Windows 8 umožňují si určitou webovou stránku připnout jako dlaždici do menu Start, tento test kontroluje, zda daná webová stránka má k dispozici tuto funkci a nabídne uživateli dlaždici s odpovídajícím obrázkem.

Testy kontrolují pouze kód stránky, takže nenahradí manuálně kontrolovanou vizuální podobu, nicméně se jedná o velmi efektivní způsob jak zkontrolovat některé funkce webu. (TechNet, 2013)

2.5.4 Browsershots

Úkolem webové aplikace Browsershots je vytváření náhledů webových stránek. V Současné době aplikace nabízí screenshoty (náhledy) z čtyř operačních systémů a přibližně padesáti verzí webových prohlížečů. Kromě rychlého náhledu je zde možnost souhrnného stažení na lokální disk. (Wagner, 2008) Tato služba je zdarma.

2.5.5 Browserstack

Tato služba nabízí kromě výše zmíněných screenshotů stránek ve všech možných prohlížečích i možnost přímo z vlastního prohlížeče testovat kompatibilitu v reálném čase. K dispozici jsou všechny běžně používané prohlížeče a dokonce i mobilní Opera, Chrome (Android) a Safari (iOS). Celkový počet možných emulací operačních systémů a webových prohlížečů je 300, což je opravdu mnoho možností v jednom nástroji. Nevýhoda je, že BrowserStack není zdarma. Nabízí tři měsíce na vyzkoušení, dlouhodobé používání stojí přibližně 20 dolarů za měsíc. Microsoft po dohodě s autory aplikace BrowserStack nabízí tří měsíční využívání Browserstacku zdarma také jako doplňku přímo do prohlížeče. (Jahoda, 2013)

2.5.6 IETester

IETester je užitečný pomocník pro testování kompatibility v prohlížeči InternetExlorer. Nabízí zobrazení od verze IE5, až po nejnovější verze programu. Vzhledem připomíná Microsoft Office 2007. V současné době je výhodnější používat nástroje, které dokážou emulovat více prohlížečů na jednom místě. (Buchta, 2013)

2.5.7 Link Checker

Po zadání adresy webu a prohledání stránky, vypíše seznam chybných odkazů. Konsorcium W3C nabízí tento nástroj volně dostupný online. Nefungující odkazy mohou odradit návštěvníky nebo potenciální zákazníky, proto je tato kontrola důležitá. (Buchta, 2013)

2.6 Testovací tým

Tester je obecné označení pro člověka, jež se zabývá testováním. Je nutné, aby byl tester důkladně seznámen s aplikací a nastudoval její specifikaci. Veškeré nalezené chyby a nejasnosti hlásí do testovacího protokolu a předává k opravení vývojovému týmu. Správný tester zpochybňuje produkt, dokud si jej sám neověří několika praktickými příklady.

Cíl testerů definuje Ash v (2003, str. 4): „*The whole point of a tester or a test organization is to find problems and to bring them to appropriate amount of attention*.“

Tento člověk musí disponovat kritickým myšlením, je vynalézavý, pečlivý, a je schopen nalezené chyby správně interpretovat týmu. V oblasti testování se člověk stále učí. Pokud bychom se zajímali o to, co mají dobří testeři společného, pak následující vlastnosti byly společné u všech zkoumaných testerů s vysokou výkonností:

- zkušenost s produktem,
- zkušenost v dané byznys oblasti, doménová znalost,
- zkušenost s programováním,
- zkušenost se speciálními testovacími technikami,
- psaní dobrých hlášení chyb,
- udržení nadhledu/přehledu,
- nadšení z práce – skutečně je baví,
- pečlivost, svědomitost, trpělivost a výdrž. (Borovcová, 2009)

„*Dobrý tester je schopný dívat se na systém jako programátor, koncový uživatel a analytik zároveň, což vyžaduje kombinaci nejen technických, ale i řady jiných (typicky sociálních) dovedností, které často zúročí na vyšších pozicích – například při řízení týmu či vyjednávání se zákazníkem*.“ (Havlíčková, Roudenský, 2013, str. 55)

Hutcheson (2003, str. 16) komentuje úroveň studia testování na vysokých školách ve Spojených státech: “*Málo univerzit nabízí semináře testování softwaru. Dokonce ještě méně jich vyžaduje testování softwaru jakou součást vzdělávání v softwarovém inženýrství. Naneštěstí tím zasílají vzkaz byznysu a vývojářské komunitě, že testování nestojí za to*.“

Testovací tým se v ideálním případě skládá z více pracovníků. V tomto týmu je spektrum rolí, které je dáno firemní kulturou a také používanou metodikou.
Nejčastější role z hlediska hierarchie jsou:

- tester (junior/senior),
- tester analytik,
- tester manažer.

Tester provádí zpravidla manuální testy podle předem připravených testovacích scénářů od testera analytika. Analytik je člověk, který připravuje dokumenty a podklady pro testování.

Tester manažer vytváří testovací strategii pro daný projekt. Tato strategie obsahuje požadavky, plán, harmonogram, etapy testů, metriky hodnocení a výstupy testování.

Kromě rozdělení z hlediska hierarchie, existuje také rozdělení z hlediska specializace. Tester se podle své role specializuje na testování konkrétní věci:

- tester designu,
- tester funkčnosti,
- tester použitelnosti a přístupnosti,
- tester bezpečnosti a výkonu. (Borovcová, 2009)

2.6.1 Odpovědnost testera

Tester nese odpovědnost za kvalitu výsledného produktu, proto je důležité, aby člověk, který jako tester pracuje, uměl tuto odpovědnost přijmout. Je nutné, aby s určitou mírou pravděpodobnosti potvrdil, že produkt funguje tak, aby bylo možné jej předat klientovi. Všechny chyby by měly být řádně a včas reportovány. Základním axiomem testování je však tvrzení, že nelze nikdy dokázat neexistenci chyb, vzhledem k tomu, že je nemožné otestovat celý produkt. Výjimkou jsou extrémně triviální programy. Dijkstra (1972) tvrdí, že: *„…testování programu může být velmi efektivní způsob, jak ukázat přítomnost chyb, ale je beznadějně nevhodné k prokázání jejich nepřítomnosti.“*

Faktem je, že předaná aplikace někdy obsahuje chyby. Veškeré chyby jsou brány jako selhání testera nikoli programátora. Tester může argumentovat nedostatkem času na testování nebo nedostupností relevantních dat pro testování, ale každá tato překážka by měla být hlášena a zdokumentována včas. Testerovou zodpovědností je také dohlédnout na porovnání specifikace s produktem a na řádné opravení chyb, k čemuž je nutný určitý stupeň asertivity. (Borovcová, 2009)

2.6.2 Reportování chyb

Srozumitelné reportování nalezených chyb je nedílnou součástí práce testera. Správně napsaný report může urychlit nalezení a odstranění chyb, zatímco nevhodně napsaný report se bude přesouvat mezi testerem a vývojářem tak dlouho, dokud nebude vývojáři vše jasné. Hlášení chyb by mělo být stručné, přesné a poskytovat co nejvíce informací, které urychlí hledání chyby při ladění. Základním účelem hlášení je tedy poskytnout vše potřené k tomu, aby byla chyba co nejrychleji a správně opravena. (Borovcová, 2009)

2.6.3 Odhad pracnosti

Odhadování pracnosti testování je důležitým vstupem pro plánování celého projektu. Základní otázkou je, o jak velký projekt se jedná. Odhadování je ne vždy přesný způsob zjištění objemu projektu, na který působí mnoho faktorů. Pro odhad slouží několik metod:

- expertní odhad – odhad specialisty na základě zkušeností s obdobným projektem,
- metoda založená na podílech – odhad podílu z celkového vývoje a výpočet podílu,
- rozdělení celkového času – rozdělení pevného času a zdrojů v rámci testování,
- extrapolace předchozích dat – odhad na základě předchozího průběhu,
- odhad funkčních bodů – odhad velikosti testovaných částí systému,
- odhad pomocí seznamu testovacích případů.

Co může způsobit odchylku od odhadovaného času: nesprávně použitá metoda, velký počet chyb, špatná specifikace, pozdní dodávky subsystémů, nefunkční prostředí nebo špatná komunikace v týmu. Do odhadu pracnosti musí být zahrnut také čas na opravu chyb. V případě, že by čas na opravu zahrnut nebyl, každá oprava by znamenala nárůst ve zpoždění vývoje projektu. (Bureš, 2013) Každý projektový manažer musí počítat s určitou rezervou při plánování projektu. Pozdní dodávka softwaru, případně chybovost je potrestána pokutou plynoucí ze smlouvy se zákazníkem, proto je odhadování pracnosti tak důležité.

2.6.4 Certifikace ISTQB

V oboru testování softwaru je nejznámější certifikací ISTQB. Tato organizace si klade za cíl vytvořit mezinárodně uznávanou skupinu specialistů v oblasti testování softwaru. Zkoušky lze vykonávat v českém i anglickém jazyce a jsou ve formě testových otázek. K absolvování nejnižší úrovně certifikace je třeba středoškolské vzdělání a praxe v oblasti IT. Tato organizace má také českou odnož, která se stará o školení a propagaci profese testování. (Hlava, 2013) Certifikát je tedy celosvětovým měřítkem odborníků na testování a na životopisu uchazeče o pozici testera se rozhodně vyjme.

2.7 Oblast využití implementace

2.7.1 Charakteristika organizace

Prvním krokem ke způsobilosti, jakožto malý nebo střední podnik, je nutné být pokládán za podnik. Podle nové definice Komise ES (2008) je podnikem: „*každý subjekt vykonávající hospodářskou činnost, bez ohledu na jeho právní formu. K těmto subjektům patří zejména osoby samostatně výdělečně činné a rodinné podniky vykonávající řemeslné či jiné činnosti a obchodní společnosti nebo sdružení, která běžně vykonávají hospodářskou činnost*.“ Srovnání vašich údajů s prahy pro tato tři kritéria vám umožní určit, zda jste mikro podnikem, malým nebo středním podnikem:

- počet zaměstnanců,
- roční obrat,
- bilanční suma roční rozvahy (velikost aktiv).

Údaje, které se mají použít pro stanovení počtu zaměstnanců a finančních veličin, jsou údaje vztahující se k poslednímu uzavřenému zdaňovacímu období vypočtené za období jednoho kalendářního roku. Malé podniky jsou vymezeny jako podniky, které zaměstnávají méně než 50 osob a jejichž roční obrat nebo bilanční suma roční rozvahy nepřesahuje 10 milionů eur.

Počet zaměstnanců je rozhodujícím počátečním kritériem k určení, do které kategorie podnik patří. Vztahuje se na osoby s plným pracovním úvazkem, částečným pracovním úvazkem a sezónní pracovníky a zahrnuje:

- zaměstnance,
- osoby pracující pro podnik v podřízeném postavení, které jsou považovány za zaměstnance v souladu s vnitrostátním právem,
- vlastníky, kteří řídí společnost,
- společníky zapojené do běžné činnosti podniku, kteří využívají finančních výhod plynoucích z podniku.

Roční obrat se určuje výpočtem příjmů, které váš podnik získal během daného roku z prodeje a ze služeb po odečtení vyplacených slev. Obrat by neměl zahrnovat daň z přidané hodnoty (DPH) ani jiné nepřímé daně. Bilanční suma roční rozvahy se vztahuje k hodnotě hlavních aktiv vaší společnosti. (Komise ES, 2008)

Hlavní výhody malých a středních podniků podle (Novotný, Suchánek, 2004) jsou:

- pružné reagování na jakékoliv změny,
- inovativnost,
- vytváření nových pracovních příležitostí,
- odolnost proti hospodářské recesi,
- rychlost přijímání podnikatelských rozhodnutí.

Jako hlavní nevýhody jsou uvedeny:

- omezené možnosti zaměstnávání odborníků ve správě a řídících činností,
- vyšší intenzita práce a méně příznivé pracovní podmínky,
- omezené možnosti získávání výhod z rozsahu produkce,
- omezené prostředky na propagaci a reklamu.

2.7.2 Management malé firmy

Řízení malých firem je v mnoha ohledech specifické. Operativní řízení zde převládá nad strategickým a ústní komunikace jednoznačně nad komunikací psanou. Vzhledem k malému počtu zaměstnanců i vedoucích pracovníků dochází k rozdělení některých funkcí do kompetence několika málo pracovníků, často je to jeden až dva lidé. Malé firmy se neobejdou bez firemní strategie a strategie organizace práce a času. Strategie zahrnuje rozhodnutí o poskytování produktů a služeb, určení cílového trhu, rozhodnutí o stylu vedení zaměstnanců apod.

Stanovení cílů firmy je jeden ze základních kamenů úspěchu a vize je výhled do budoucna, který má hnát firmu vpřed. Malé firmy mají schopnost lépe se přizpůsobit inovacím a změnám v organizaci firmy a také pracovním technologiím. Ve firmě se ztrácí anonymita, práce je založena na přímém kontaktu s vedením a vzájemnou důvěrou. Výhodou je také možnost zaměstnanců zapojit se do dění ve firmě a přispívat svými tvůrčími schopnostmi.

Nevýhody v oblasti financí plynou především z omezení finančních zdrojů a to především u individuálních podnikatelů. Hlavním zdrojem financování je samofinancování. Jinou možností jsou podíly dalších podílníků – zde ovšem hrozí omezení práva podnikatele. Nejdůležitějším zdrojem cizího kapitálu jsou bankovní úvěr a dodavatelský úvěr. Snahou každého podnikatele je dosažení zisku a maximalizace tržní hodnoty podniku. Dalším ukazatelem je ziskovost, tedy poměr zisku k vynaloženým prostředkům. Všechny tyto

informace jsou dále sledovány a porovnávány také s minulým obdobím, aby bylo zřejmé, jak si podnik vede. (Srpová, Řehoř a kolektiv, 2010; Popesko, 2009)

Za úspěšný můžeme považovat projekt, který splnil:

- účel,
- očekávání,
- cíle projektu,
- ostatní klíčové parametry projektu.

„*Účel je důvod, proč má smysl pro podnik investovat prostředky, zdroje, čas a pracovní úsilí do realizace projektu. Vyjadřuje konkrétní a specifický přínos celé firmě, výraznou přidanou hodnotu k současnému stavu.*“ (Komzák, 2013, str. 30)

3 Analýza současného stavu

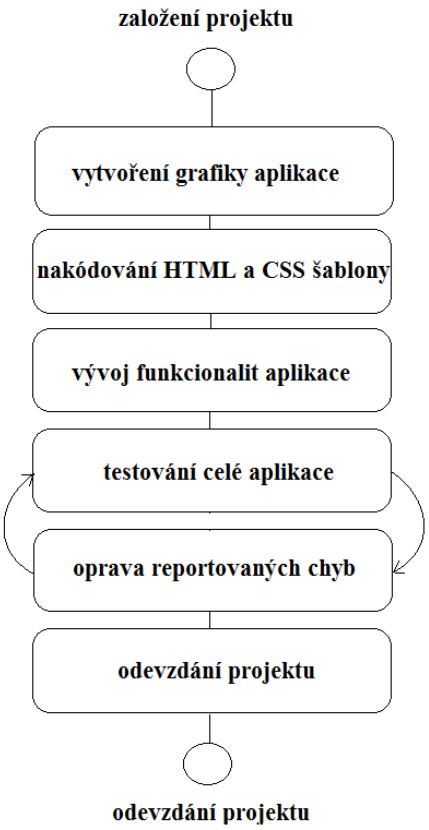
Záměrem této práce je mimo jiné návrh inovace testovacích procesů malé firmy. Před samotným navržením je třeba zanalyzovat současný stav procesů v konkrétní firmě a získat tak informace pro návrhy možných řešení.

3.1 Objekt inovace

Firma, které se inovace týká, patří do podnikatelské kategorie s méně než 50 zaměstnanci - malá firma. Firma je společností s ručením omezeným. Její hlavní činností je vývoj webových aplikací, přičemž většina těchto aplikací jsou internetové obchody s individuálními požadavky. Dalšími oblastmi zájmu jsou reklamní grafika, internetový marketing, propojení e-shopů s ERP systémy a provoz webhostingu. Klienty společnosti jsou malí podnikatelé, ale také velké firmy. Schůze vývojového týmu se konají nepravidelně, většinou dvakrát až třikrát týdně. Na schůzích jsou shrnuty úkoly za minulé dny a také úkoly nadcházející, včetně časových plánů projektů.

Současný vývoj aplikací vychází z vodopádového vývojového procesu. V procesu na obr. 3.1 je jeden cyklus, činnosti t*estování celé aplikace* a *oprava reportovaných chyb* probíhají tak dlouho, dokud testování neprojde bez chyb k reportování. Otestování aplikace až na samém konci vývojového procesu s sebou přináší řadu nevýhod. Vzhledem k tomu, že na případné fatální chyby se přijde až v pozdním stádiu celého procesu, s velkou pravděpodobností může dojít k nedodržení termínu odevzdání.

Obrázek 3.1 Proces současného vývoje



(Zdroj: vlastní)

3.2 Identifikace požadavků

Požadavky k inovaci byly identifikovány na základě open-end rozhovoru, s otevřenými otázkami, se zaměstnanci firmy. V rámci tohoto rozhovoru jsme dospěli k několika problémovým oblastem, které jsou uvedeny v následující tabulce a dále rozepsány v jednotlivých odstavcích.

Tabulka 3.1 Seznam problémových oblastí

problém	priorita	kdo problém hlásil	koho se problém týká
Špatná komunikace	vysoká	tester	celý tým
Chybějící postupy	střední	tester	tester
Nekvalitní otestování	vysoká	manažer	tester
Nekvalitní oprava chyb	vysoká	tester	vývojář
Slabá kontrola	střední	vývojář	manažer
Nedodržení termínů	vysoká	manažer	celý tým

(Zdroj: vlastní)

Špatná komunikace

Problémy v komunikaci se týkají celého týmu a mohou být příčinou mnoha nedorozumění. Je nutné, aby všichni zainteresovaní zaměstnanci měli pro ně dostatečné, informace o každém projektu, na kterém se podílí. Důležité informace o projektu a jeho specifikace musí být snadno k nalezení. Jakákoliv změna ve specifikaci musí být včas zaznamenána a členové vývojového týmu o ní musí vědět. K rozdělení úkolů na projektu má sloužit týmová schůze, k ostatním záležitostem slouží vhodný komunikační kanál nebo osobní setkání. Komunikace je základním stavebním kamenem každého projektu.

Chybějící postupy

V malých firmách často chybí výrobní postupy. V našem případě se jedná o postupy k testování. Jestliže testerovi chybí testovací scénáře, těžko může potvrdit, že otestoval „vše“, protože scénáře a postupy také slouží jako metrika testování. Tester musí jasně vědět co je cílem testování, co má testovat, kde to najde a jaké testy má provést. Po testování musí vhodně reportovat nalezené chyby a předat je vývojářům k opravení.

Nekvalitní otestování

Nekvalitní otestování má dopad hlavně na kvalitu produktu a dobré jméno firmy. Přímo však souvisí s chybějícími postupy k testování. Volné testování celé aplikace je vždy méně úspěšné, než věnování se jednotlivým typům testování zvlášť. Lepším výsledkům testování lze dosáhnout také školením, certifikací nebo přidáním dalšího testera do týmu. Zde jednoznačně platí pravidlo *více očí, více vidí*.

Nekvalitní oprava chyb

Oprava chyb není tak jednoduchá záležitost jak se může zdát. Někteří vývojáři, zvláště ti ve spěchu, chybu opraví pouze naoko, takže vypadá, že je vše v pořádku, ale při dalším procházení aplikace se chyba objeví neočekávaně znovu. Když se tento případ stane po odevzdání projektu klientovi, vzniká problém. Odpovědnost za chyby má vždy tester, ale v tomto případě za problém může nekvalitní oprava chyb vývojáře. Kvalitní a rozvážná oprava chyb může týmu ušetřit spoustu nepříjemností v budoucnu.

Slabá kontrola

Demokratický styl řízení podniku má své výhody, ale jestliže dlouhodobě nefunguje, je třeba upravit styl řízení tak, aby se zaměstnanci cítili příjemně a přitom se zpřísnila kontrola i celková organizace práce. Důsledkem slabé kontroly může být nekvalita práce a nedodržování termínů. Řešením jsou častější schůze a kontrola stavu zadaných úkolů.

Nedodržení termínů

Po sečtení všech výše zmíněných problémů dojde k nedodržení termínů odevzdání projektů, což je pro firmu velkým rizikem, kterému je zbytečné se vystavovat. Z nedodržení termínu obvykle vyplyne smluvní pokuta nebo při nejlepším nespokojenost zákazníka. Každý projekt by měl být vhodně naplánován a počítat s určitou rezervou. Odevzdání projektu se většinou prodlužuje z důvodu nečekaných změn ve specifikaci a funkčnosti projektu, ale také velmi často v případě, že se na testování a opravu chyb nevyhradí dostatek času. Pokud se nedodržování termínů stává běžnou praxí, může to mít na firmu katastrofální účinek.

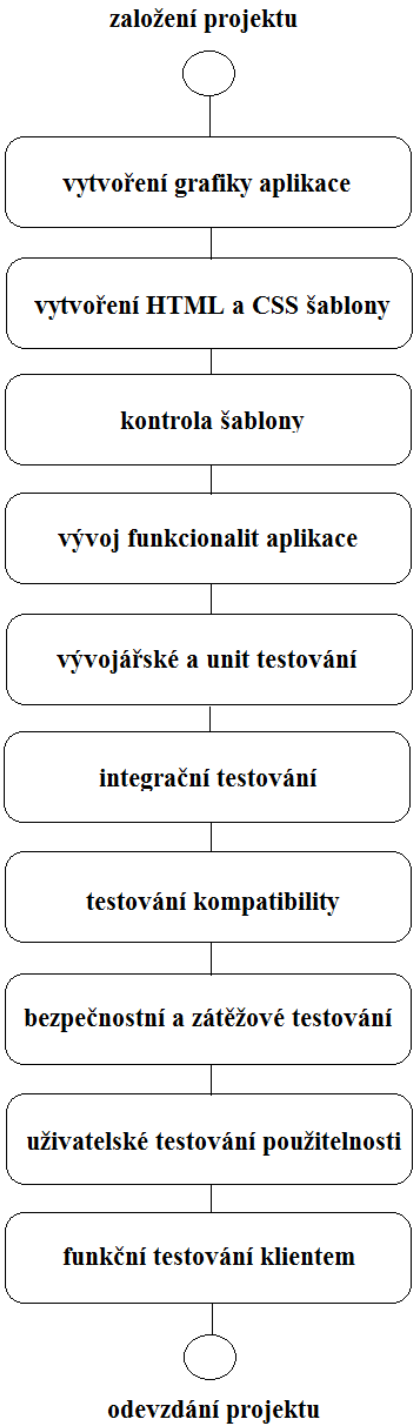
4 Návrh na inovaci a implementaci procesu testování webových aplikací

Cílem navrhovaného řešení je inovace procesů, snížení rizika a provozních nákladů. V publikaci Evropská unie a inovace je definován pojem inovace, který je popsán jako: „*soubor činností, které vedou k úspěšné výrobě, vstřebávání a využití novinek v ekonomické a sociální sféře*“. (Hronek, Prnka, Šterlink, 2002)

4.1 Navržení procesů

Životní cyklus vývoje webové aplikace je rozdělen do několika kroků. Oproti současnému procesu na obrázku 3.1, je hlavní změna v počtu kroků celého procesu. Testování je podle svého druhu rozděleno na několik jednotlivých kroků. Model je zjednodušen pro účely práce, součástí každé činnosti testování je také oprava reportovaných chyb. Jednotlivé druhy testování a oprava chyb probíhají vždy tak dlouho, dokud již nejsou reportovány žádné chyby k opravení.

Obrázek 4.1 Návrh procesu vývoje webových aplikací



(Zdroj: vlastní)

4.1.1 Specifikace testů

Při výběru testů bylo vycházeno z kapitoly 2.5. Testování, z odborné literatury a osobních zkušeností.

Kontrola vytvořené šablony

Vzhledem k tomu, že při vývoji webových aplikací je obvykle nejprve prováděna grafická část aplikace před vytvořením její funkčnosti, nabízí se zde možnost kontroly vytvořené šablony s grafickým návrhem, který je ve formě samostatných obrázků jednotlivých stránek. Šablona by měla zcela odpovídat grafickým návrhům stránek. Jakékoliv nesrovnalosti musí být prodiskutovány s kodérem, případně i grafikem a šablona se musí upravit. Kontrolu šablony může provádět jak tester, tak navíc také grafik, protože grafici jsou obvykle osoby velmi pečlivé a všímají si sebemenších detailů, zvláště když se jedná o jejich návrh.

Vývojářské a unit testování

Po vývoji funkcionalit aplikace následuje vždy vývojářské testování, kterým se programátor ujistí, že kód funguje tak jak má a může tuto část aplikace předat testerovi k otestování. Pomocí unit testování je pak testerem zkontrolována funkčnost zmíněné části aplikace. V této fázi se může přijít na chyby, které by později mohly zdržet celý proces vývoje, takže má unit testování velký význam.

Integrační testování

Integrační testování odhaluje chyby, které se projevily spojením všech částí aplikace dohromady a chyby v komunikaci s vnějším okolím aplikace. Tomuto testování je třeba věnovat zvýšenou pozornost, a pokud to kapacita zaměstnanců dovolí, zapojit do testování více testerů.

Testování kompatibility

Cílem testování je správné zobrazení aplikace v nejpoužívanějších prohlížečích a jejich různých verzích. Pro toto testování je vhodné využívat online služby BrowserStack, která dokáže nasimulovat celkem 300 kombinací operačních systémů a prohlížečů. Dalším vhodným nástrojem je modernIE, který kromě náhledů aplikace v jednotlivých prohlížečích vypíše také seznam problémů napříč prohlížeči a mnoho dalších informací, ze kterých mohou vývojáři při opravě chyb čerpat. Kromě online nástrojů k testování, by měl tester využít všech prohlížečů, které jsou nainstalované na jeho počítači, případně na dalších volných strojích, včetně mobilních zařízení, jestliže aplikace využívá responzivní design. Simulátorům se nedá stoprocentně věřit, na rozdíl od reálných zařízení.

Součástí testování kompatibility je také kontrola důležitých HTML tagů z hlediska SEO. Nutností je správně vyplněn titulek stránky, klíčová slova a popisek stránky. Obrázky by měly mít alternativní text a nadpisy postupný charakter.

Bezpečnostní a zátěžové testování

Pro zátěžové testování byl vybrán nástroj JMeter, který je zdarma a dokáže nasimulovat reálnou zátěž pro aplikaci. Nástroj má sice grafické rozhraní, ale k jeho obsluze je třeba určitá zkušenost, takže se doporučuje spíše senior testerům nebo jiným pověřeným zaměstnancům. Klíčové je objevení nedostatků při provozu aplikace a chování aplikace při selhání.

Ke kontrole bezpečnosti je navrhnut bezpečnostní skener, který vyhledává zranitelná místa v aplikaci. BackTrack Linux je bezplatný program, který nabízí mnoho bezpečnostních testů. Opět je k němu nutná určitá odborná znalost, nedoporučuje se junior testerům.

Uživatelské testování použitelnosti

Je potřebné, aby uživatelské testování prováděl člověk, který odpovídá cílové skupině aplikace, případně někdo, koho můžeme zařadit mezi běžné uživatele Internetu. V malé firmě je možné využít hned několik zaměstnanců například sekretářku, účetní, personalistu nebo obchodního zástupce. Tito uživatelé pomocí předem vytvořeného scénáře nasimulují konkrétní akce za přítomnosti testera, který si zapisuje poznámky o všech nejasnostech, které uživateli vadí. Tyto poznámky jsou pak projednány v týmu a případně jsou provedeny změny v aplikaci. K analýze použitelnosti lze využít také teplotní mapy webu. Užitečným nástrojem je SiteOverlay z Google Analytics, který zaznamenává, kde uživatelé klikají a pohybují myší, a která místa jsou naopak ignorována.

Funkční a akceptační testování klientem

Jako poslední krok před odevzdáním projektu je ověření funkčnosti aplikace podle její specifikace. Tester simuluje chování koncového uživatele a samotný klient si celou aplikaci projde, aby ověřil, že aplikace funguje podle jeho představ. Jestliže klient dojde k závěru, že je vše správně, může se projekt odevzdat. V případě, že je nutná úprava specifikace, obvykle je nutné upravit také grafický návrh a celou funkčnost, což znamená zopakovat celý proces testování.

4.1.2 Využití nástroje Basecamp

Basecamp je aplikace pro podporu managementu v malých a středních firmách. Jedná se o komerční nástroj, jehož provoz se odvíjí od potřebného počtu projektů a velikosti úložiště. Za 1000 korun měsíčně nabízí Basecamp správu až 40 projektů a 15GB využitelného místa. Vlastnosti nástroje jsou:

- sledování projektů odkudkoliv,
- zobrazení nejnovějších zpráv o každém projektu,
- přepínání mezi diskuzemi a možnost psát okamžitě své myšlenky,
- měření času,
- rozesílání zpráv a komentářů,
- okamžité zobrazení pokroků, když členové projektového týmu nějaký vloží,
- zobrazení veškerých materiálu k projektům,
- zabezpečený přístup a využití uživatelských práv.

Basecamp podporuje přehledné plánování, zrychlení a přehlednost komunikace a informovanost o aktuálním dění. Tester vloží seznam chyb do tzv. To-do listu (volně přeloženo jako seznam věcí co se má udělat), má možnost připojit obrázky a při opravování chyb vývojář jednoduše tlačítkem zkompletuje konkrétní chyby. O kompletaci se mohou okamžitě všichni dovědět na nástěnce, tester tak může ihned překontrolovat, jestli je chyba správně opravena. Do Basecampu mohou mít přístup také klienti, stačí pro ně vhodně nastavit práva. Klienti mohou diskutovat nebo vkládat dokumenty a jsou informování o stavu projektu.

4.2 Vyhodnocení návrhu

Vzhledem k tomu, že stav implementace inovace procesů testování je ve fázi příprav, zatím není možné změřit ukazatele přínosu. Jak velký přínos bude tato inovace mít, můžeme pouze odhadovat, ale veškeré postupy v této práci byly metodicky vedeny tak, aby se dosáhlo lepších výsledků. Po zavedení navrhovaných změn proběhne kvalitativní zhodnocení celé inovace formou rozhovoru s vedením a pracovníky firmy. Navrhované změny jsou hlavně změnou struktury dosavadních procesů. Automatizace testování zde byla využita jen pro zátěžové testování a součástí opravování chyb se předpokládá průběžný refaktoring.

Samotné nasazení nových postupů se vždy projeví mírným poklesem produktivity v krátkodobém časovém období. Následně se očekává postupný nárůst produktivity

v dlouhodobém hledisku. Náročnost na finanční prostředky vzroste díky využití komerčních nástrojů Basecamp a Browserstack, ale celkové náklady se pravděpodobně dlouhodobě sníží díky kvalitnějšímu testování a vhodnému nástroji na podporu managementu.

5 Závěr

Cílem práce bylo představit webové aplikace a metody jejich vývoje, popsat proč je důležité testovat, jaké testy existují a zpracovat návrh možné metodiky pro testování aplikací v malých firmách.

Ke splnění cíle bylo provedeno shrnutí teoretických východisek vývoje a testování, kterým se věnuje první kapitola. V další kapitole pak analýza současného stavu testování v malé firmě, včetně jejich slabých stránek. Na závěr byl vypracován návrh na inovaci samotného procesu testování, kde jsou popsány jednotlivé testy a využití nástroje pro podporu řízení projektů.

Při návrhu metodiky testování bylo vycházeno z analýzy aktuálních procesů firmy a teoretických poznatků nabytých při studiu několika odborných knih a článků, v českém i anglickém jazyce. V rámci analýzy byly identifikovány problémové oblasti, které měla inovace zohlednit. V návaznosti na jednotlivé problémy byla představena možná řešení. Jednotlivých dílčích cílů bylo v průběhu práce dosaženo, stejně jako došlo k naplnění cíle hlavního. Součástí návrhu je také vyhodnocení inovace, které popisuje očekávané důsledky z hlediska produktivity a nákladů.

Práce nabízí srozumitelný rozsah informací o webových aplikacích, vývoji software a testování, které mohou pomoci jak nezkušeným uživatelům, kteří se s touto problematikou ještě nesetkali, tak testerům a ostatním členům vývojového týmu, k pochopení smyslu testování. V praktické části je navrhnuta metodika testování, od které se očekává budoucí přínos ve formě zvýšení efektivity a snížení nákladů. Samotná implementace ve firmě teprve proběhne.

V budoucnu by se měl celý vývoj webových aplikací ve firmě ubírat směrem agilních metodik, konkrétně řízení vývoje pomocí metodiky Scrum. Do testování více zahrnout možnost automatizace testů a provádět pravidelně refaktoring, který ulehčí budoucí změny v aplikaci.

6 Seznam použité literatury

Literatura

ASH, Lydia. The Web testing companion: the insider's guide to efficient and effective tests. Indianapolis: Wiley, 2003, 554 s. ISBN 04-714-3021-8.

BECK, Kent. *Extrémní programování*. Vyd. 1. Praha: Grada, 2002, 158 s. ISBN 80-247-0300-9.

BECK, Kent. *Programování řízené testy*. 1. vyd. Praha: Grada, 2004, 203 s. Moderní programování. ISBN 80-247-0901-5.

BOEHM, Barry and Papaccio N. Understanding and Controlling Software Costs. In IEEE Transactions on Software Engineering. TRW Inc., Redondo Beach, CA., 1988. ISSN 0098-5589.

BUCHALCEVOVÁ, Alena. *Agilní metodiky*. Praha 14.11.2002 – 15.11.2002. In: Objekty 2002. Praha : ČZU, 2003, s. 53–61. ISBN 80-213-0947-4.

FOWLER, Martin. *Refactoring: Zlepšení existujícího kódu*. Praha: Grada, 2003, 394 s. ISBN 80-247-0299-1.

HAVLÍČKOVÁ, Anna a Petr ROUDENSKÝ. *Řízení kvality softwaru: Průvodce testováním*. Brno: Computer Press, 2013. ISBN 978-80-251-3816-8.

JORGENSEN, Paul C. *Software testing: a craftsman´s approach*. 3. vyd. Boca Raton: Auerbach Publications, 2008. ISBN 978-0-8493-7475-3.

KADLEC, Václav. *Agilní programování: metodiky efektivního vývoje softwaru*. 1. vyd. Brno: Computer Press. ISBN 80-251-0342-0.

KOMZÁK, Tomáš. *Řízení IT projektů pro úplné začátečníky*. 1. vyd. Brno: Computer Press, 2013, 213 s. ISBN 978-80-251-3791-8.

LAZARIS, Louis, Alexis GOLDSTEIN a Estelle WEYL. *HTML5 & CSS3 For The Real World*. Collingwood, VIC, Australia: SitePoint, 2011. ISBN 978-0-9808469-0-4.

MCCONNELL, Steve. *Rapid Developement: Taming Wild Software Schedules*. Microsoft Press, 1996, 72 s. ISBN 1-55615-900-5.

NOVOTNÝ, J. a SUCHÁNEK, P. *Nauka o podniku I*. 1. vyd., Brno: Masarykova univerzita, 2004, 184 s. ISBN 80-210-3333-9.

PATTON, Ron. *Testování softwaru*. Praha: Computer Press, 2002, 313 s. Programování. ISBN 80-722-6636-5.

POPESKO, Boris. *Moderní metody řízení nákladů: jak dosáhnout efektivního vynakládání nákladů a jejich snížení*. 1. vyd. Praha: Grada, 2009, 233 s. Prosperita firmy. ISBN 978-80-247-2974-9.

PRNKA, Tasilo, František HRONEK a Karel ŠPERLINK. *Inovace v Evropské unii.* 2., aktualiz. vyd. Ostrava: Repronis, 2003, 81 s. ISBN 80-732-9042-1.

ŘEPA, Václav. *Podnikové procesy: procesní řízení a modelování*. 2., aktualiz. a rozš. vyd. Praha: Grada, 2007, 281 s. ISBN 978-80-247-2252-8.

SMIČKA, Radim. *Optimalizace pro vyhledávače - SEO: jak zvýšit návštěvnost webu*. Vyd. 1. Kralice na Hané: Grada, 2004, 126 s. Moderní programování. ISBN 80-239-2961-5.

SRPOVÁ, Jitka a Václav ŘEHOŘ a kolektiv. *Základy podnikání: teoretické poznatky, příklady a zkušenosti českých podnikatelů*. 1. vyd. Praha: Grada, 2010, 427 s. ISBN 978-80-247-3339-5.

YOURDON, Edward. *Classics in software engineering*. Englewood Cliffs, NJ: Yourdon, 1979. ISBN 01-313-5179-6.

Zdroje z internetu

AGILEMANIFESTO: Manifest Agilního vývoje software. [online] 2001. [cit. 16.2.2013]. Dostupné z: http://www.agilemanifesto.org/iso/cs/

BOROVCOVÁ, Anna. *Testování webových aplikací* [online]. Praha, 2008 [cit. 10.4.2014]. Dostupné z: http://testovanisoftwaru.blogspot.cz/p/dptestovanisoftwarupdf.html. Diplomová práce. Univerzita Karlova v Praze.

BUCHTA, Martin. *IETester* [online]. 2013 [cit. 16.4.2014]. Dostupné z: http://preklady.buchtic.net/ietester#.U0_oWlV_srV

BUCHTA, Matin. *Freeware: LinkChecker prověří odkazy na vašem webu* [online]. 2013 [cit. 17.4.2014]. Dostupné z: http://pcworld.cz/downloads/freeware-linkchecker-proveri-odkazy-na-vasem-webu-46720

BUREŠ, Miroslav. *Základy testování software: Metody pro odhadování pracnosti testů, ekonomika a efektivita testování*. 2013. Dostupné z: http://webdev.felk.cvut.cz/~buresm3/ts1/TS1_prednaska12.pdf

DANĚK, Antonín. *Firebug - mocný nástroj pro Firefox: nejen pro webové vývojáře* [online]. 2007 [cit. 14.4.2014]. Dostupné z: http://blog.antonindanek.cz/clanek/firebug-mocny-nastroj-pro-firefox-nejen-pro-webove-vyvojare/

FI MUNI. *Test-Driven Development — programování řízené testy* [online]. 2005 [cit. 1.4.2014]. Dostupné z: http://www.fi.muni.cz/usr/jkucera/pv109/2005/xvlcek1.htm

GAVIN, Ryan. MICROSOFT. *Internet Explorer 10 now available for more than 700M Windows customers* [online]. 2013 [cit. 10.4.2014]. Dostupné z: http://blogs.windows.com/ie/b/ie/archive/2013/02/26/internet-explorer-10-now-available-for-more-than-700m-windows-customers.aspx

HAVRLANT, Lukáš. *Sémantika: pravý význam HTML značek* [online]. 2005 [cit. 14.4.2014]. Dostupné z: http://semantika.name/

HLAVA, Tomáš. *Testování softwaru* [online]. 2013 [cit. 17.4.2014]. Dostupné z: http://testovanisoftwaru.cz

CHAPMAN, James. *Software Developement Methodology* [online]. 2004 [cit. 17.4.2014]. Dostupné z: http://www.hyperthot.com/pm_sdm.htm

JAHODA, Bohumil. *Internet Explorer 11* [online]. 2013 [cit. 8.4.2014]. Dostupné z: http://jecas.cz/ie11

JANOVSKÝ, Dušan. *Jak psát web: o tvorbě internetových stránek* [online]. 2014 [cit. 16.4.2014]. Dostupné z: Jakpsatweb.cz

KADLEC, Václav. *Extremismus nemusí být na škodu: extrémní programování* [online]. 2003 [cit. 3.3.2014]. Dostupné z: http://www.zive.cz/clanky/extremismus-nemusi-byt-na-skodu-extremni-programovani/sc-3-a-111714/default.aspx

KOCOUREK, Zdeněk. *Procesní řízení v organizaci* [online]. 2007 [cit. 5.4.2014]. Dostupné z: http://modernirizeni.ihned.cz/c1-22611310-procesni-rizeni-v-organizaci

KOSEK, Jiří. *Vše o WWW* [online]. 2013 [cit. 17.4.2014]. Dostupné z: http://www.kosek.cz/

KRATOCHVÍL, Jiří. *Řízení vývoje – Metodika Scrum* [online]. 2010 [cit. 5.4.2014]. Dostupné z: http://jiri.kratochvil.eu/rizeni-vyvoje-metodika-scrum/

KÜMMEL, Roman. SEZNAM.CZ. *Bezpečnost webových aplikací a její testování* [online]. 2014 [cit. 5.4.2014]. Dostupné z: http://www.slideshare.net/cCuMiNn/webove-zranitelnosti?ref=http://vyvojari.seznam.cz/akce/146-bezpecnost-webovych-aplikaci-jeji-testovani

LÁNG, Peter. *Co je to framework?* [online]. 2010 [cit. 17.4.2014]. Dostupné z: http://langi.cz/webarna/tag/frameworky

LUKEŠ, Pavel. *Zátěžové testování webových aplikací - úvod* [online]. 2014 [cit. 5.4.2014]. Dostupné z: https://www.etnetera.cz/773-tech_life/tech_life_140117_zatezove_testovani_webovych_aplikaci.html

MAJDA, David. *Knihovny vs. frameworky* [online]. 2009 [cit. 17.4.2014]. Dostupné z: http://majda.cz/zapisnik/265

MAREŠ, Jan. *Jak vytvořit responzivní design webu* [online]. 2013 [cit. 16.3.2014]. Dostupné z: http://www.whitehat.cz/jak-vytvorit-responzivni-design/

MICHÁLEK, Martin. *Jak testovat responzivní weby* [online]. 2013 [cit. 16.4.2014]. Dostupné z: http://www.vzhurudolu.cz/prirucka/jak-testovat-responzivni-weby

PÁRAL, Tomáš. *Testování výkonu webových aplikací* [online]. 2010 [cit. 4.4:2014]. Dostupné z: http://vsadnajavu.cz/2010-02/nezarazene/testovani-vykonu-webovych-aplikaci/

PAVLÍČEK, Radek. *Přístupný web a jak se vyvarovat chyb* [online]. 2009 [cit. 14.4.2014]. Dostupné z: http://www.mvcr.cz/clanek/pristupny-web-a-jak-se-vyvarovat-chyb.aspx

SEDLÁČEK, Miroslav. *Demingův cyklus PDCA* [online]. 2011 [cit. 3.4.2014]. Dostupné z: http://www.systemonline.cz/sprava-it/deminguv-cyklus-pdca.htm

STOHWASSER, Petr. *Pěstujeme web* [online]. 2010 [cit. 9.4.2014]. Dostupné z: http://www.pestujemeweb.cz/

ŠARMANOVÁ, Jana. VŠB. *Databázové a informační systémy* [online]. 2007 [cit. 3.4.2014]. Dostupné z: http://www.elearn.vsb.cz/archivcd/FEI/DAIS/DAIS.pdf

ŠTRUPL, František. H1.CZ. *Použitelnost webu* [online]. 2010 [cit. 5.4.2014]. Dostupné z: http://www.slideshare.net/h1cz/pouitelnost-webu

TECHNET. *Testování kompatibility webových stránek – Modern.IE* [online]. 2013 [cit. 10.4.2014]. Dostupné z: http://blogs.technet.com/b/technetczsk/archive/2013/02/04/testovani-kompatibility-webovych-stranek-modern-ie.aspx

ÚVT MUNI. *SCRUM* [online]. 2013 [cit. 2.3.2014]. Dostupné z: http://www.muni.cz/ics/925600/web/scrum

W3C. *HTML5: A vocabulary and associated APIs for HTML and XHTML* [online]. 2013 [cit. 17.4.2014]. Dostupné z: http://www.w3.org/TR/html5/

W3SCHOOLS. *Browser Statistics and Trends* [online]. 2014 [cit. 10.3.2014]. Dostupné z: http://www.w3schools.com/browsers/browsers_stats.asp

WAGNER, Janek. *BrowserShots: Otestujte váš webdesign v různých prohlížečích* [online]. 2008 [cit. 15.4.2014]. Dostupné z: http://blog.lupa.cz/jankuv/browsershots-otestujte-vas-webdesign-v-ruznych-prohlizecich/

7 Seznam zkratek

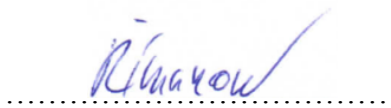
CSS	Cascading Style Sheets *(Kaskádové styly)*
ERP	Enterprise resource planning *(Systém pro řízení firemních zdrojů)*
FDD	Feature Driven Developement (*Vývoj řízený užitnými vlastnostmi daného software*)
HTML	Hyper Text Markup Language (*Značkovací jazyk pro hypertext)*
HTTP	Hypertext Transfer Protocol *(Protokol pro přenos hypertextu)*
IE	Internet Explorer
ISTQB	International Software Testing Qualifications Board *(Mezinárodní softwarová rada testování)*
IT	Information Technology *(Informační technologie)*
PDCA	Plan Do Check Act *(Plánuj, proveď, zkontroluj, konej)*
RUP	Rational Unified Process *(Unifikovaný proces vývoje softwaru)*
SEM	search engine marketing *(Marketing pro vyhledávače)*
SEO	Search engine optimalization *(Optimalizace pro vyhledávače)*
SW	Software
TDD	Test driven developement *(Vývoj řízený testy)*
UML	Unified Modeling Language *(Unifikovaný modelovací jazyk)*
URL	Uniform resource locator *(Jednotný lokátor zdrojů)*
W3C	World Wide Web Consortium *(Konsorcium celosvětové sítě)*
WWW	World-wide web *(Celosvětová síť)*
XP	Extreme Programming *(Extrémní programování)*

Prohlášení o využití výsledků bakalářské práce

Prohlašuji, že

- jsem byla seznámena s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo;
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, bakalářskou práci užít (§ 35 odst. 3);
- souhlasím s tím, že bakalářská práce bude v elektronické podobě archivována v Ústřední knihovně VŠB-TUO a jeden výtisk bude uložen u vedoucího bakalářské práce. Souhlasím s tím, že bibliografické údaje o bakalářské práci budou zveřejněny v informačním systému VŠB-TUO;
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona;
- bylo sjednáno, že užít své dílo, bakalářskou práci, nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Ostravě dne 8. 5. 2014



.....

Tereza Římanová